Domain Decomposition Approaches for the Generation of Equidistributing Parametric Curves

by

© **Miranda Boutilier**

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science.

Scientific Computing Program

Memorial University

August 2021

St. John's, Newfoundland and Labrador, Canada

Abstract

Moving mesh methods are often used to solve boundary value problems whose solutions contain regions of rapid change. In this case, these moving mesh methods allow us to concentrate a fixed number of nodes in these regions of high variance. These meshes are obtained by solving a second order boundary value problem (BVP), which arises from the equidistribution principle.

There are many real-world examples where boundary value problems are posed on curves and surfaces. Here, we focus on the case where the problem is posed on a curve that is able to be explicitly represented parametrically as $\mathbf{x} = (x_1(r), x_2(r), \ldots, x_n(r)) \in \mathbb{R}^n$. When the solution has regions of rapid change or the curve on which the problem is posed has regions of high variance or curvature, moving mesh methods allow us to find a mesh that better resolves the function on the curve without adding additional nodes. We consider combining the solution of mesh equations with the solution of differential equations posed on parametric curves. These differential equations include both time-dependent partial differential equations (PDEs) and time-independent boundary layer problems.

In addition to considering the above on a single domain, we extend these methods to form multi-domain iterations to solve these boundary value problems. Domain decomposition allows us to harness the power of parallel computing, a topic that has become popular in recent years with the increase of computing power. We provide multi-domain iterations for both time-dependent and time-independent differential equations posed on parametric curves; these include classical Schwarz and optimized Schwarz methods. These iterations are formed such that they are able to be performed in parallel. Numerical results are provided throughout to illustrate the results of the iterations. This thesis also includes theoretical results that generalize known results for classical Schwarz and optimized Schwarz methods to the case where the problem is defined on a curve.

Acknowledgements

I would first like to thank my supervisor, Dr. Ronald Haynes, for his guidance and patience during my studies at Memorial University of Newfoundland. The consistent direction and encouragement during my studies was imperative to the completion of this thesis. I am extremely grateful for the above and beyond support.

I would like to thank the School of Graduate Studies and Scientific Computing program for their financial support during my studies. I would also like to thank the Department of Mathematics and Statistics for the arrangement of my Teaching Assistantship and additional work opportunities. During the stress of the pandemic, it was crucial to know that their financial support was unwavering.

I am grateful to the faculty and staff of the Department of Mathematics and Statistics for their instruction and assistance throughout my time at Memorial University of Newfoundland. I would also like to thank the members of our research group for their consistent inspiration and ideas.

Lastly, I would like to thank my parents, sister, and friends for their constant support and encouragement.

Table of contents

List of tables

List of figures

Chapter 1

Introduction

Mathematical models in a variety of scientific areas (fluid mechanics, mathematical biology, engineering, finance, etc.) involve the solution of partial differential equations (PDEs). In practice, one is typically not able to solve these PDEs exactly. Therefore, numerical methods are often used to approximate the true solutions. Most numerical methods involve the partitioning of the spatial and temporal domains on which the PDE is defined. The set of nodes that make up the spatial partitioning is referred to as a mesh.

Non-uniform meshes are often used to obtain the efficient numerical solution of PDEs. Often, a uniform mesh can miss certain features of a solution, and adapting the mesh is a way to solve this problem. The choice of mesh generally stems from the equidistribution principle. In this thesis, we focus on r-refinement, a mesh refinement method that keeps a constant number of nodes and relocates them to increase the resolution of a function. In the case of time-dependent problems, these are often referred to as moving mesh methods [31].

The integral formulation of the equidistribution principle (EP) was first introduced by de Boor [7]. In this principle, we look to relocate mesh points to make some function of the solution uniform, or "equidistributed", with respect to the mesh. For this, we wish to have the area under a given function $M(x) > 0$ to be equal over each subinterval of the mesh. Here, this function $M(x)$ determines the mesh for the function of interest $u(x)$ and is typically dependent on $u(x)$; that is, $M(x) = M(u(x))$. This function $M(x)$ is referred to as a monitor function.

We choose a monitor function $M(x)$ that is representative of the difficulty in resolving $u(x)$; that is, the value of $M(x)$ increases at x values for which $u(x)$ is difficult to resolve. Since we expect the error of the numerical solution to be large in the areas where M is large, then the equidistribution principle will result in concentrated nodes in these areas.

These moving mesh methods can also be applied to both the equidistribution of parametric curves and the equidistribution of functions posed on static curves. In the case of equidistribution of a parameterized curve $(x(r), y(r))$ for $r \in [0, 1]$, we use the EP and a monitor function to determine our new non-uniform grid r. Here, the monitor function uses the curve features to determine where to concentrate the nodes. This is described in detail in [39]. In the case of equidistribution of a known analytical function $u(r)$ posed on $(x(r), y(r))$, we use monitor functions that consider both the function and curve features to equidistribute. Multiple monitor functions are compared by interpolation errors from the fine numerical solution. Experimenting with monitor functions in the case where u is known is beneficial, as it gives us information on appropriate monitor functions that can be used in the case where u is unknown.

We can also apply similar techniques to the case of differential equations posed on a parametric curve, where u is the unknown solution of a differential equation. PDEs describe a large number of real-world applications and are often not posed on a line, but rather on a curve or surface. These problems arrive in biology, fluid dynamics, neuroscience, physics, and other areas. In biology, lipid bilayer membranes (biomembranes) can be treated as fluid surfaces [10, 40]. In fluid dynamics, surfactants on fluid-fluid interfaces can be modeled as a PDE on a moving hypersurface [1, 34]. If the geometry of the physical problem can be considered "thin" in some direction, we can simplify the model to a PDE on a lower dimensional curve [35]. This can occur in thin shells, cell membranes, butterfly wings, animal coats, and other thin surfaces. For an overview of applications in mathematical biology, see [42]. In this thesis, we propose algorithms to solve both static boundary layer problems and time-dependent PDEs posed on curves.

To solve static boundary layer problems, we employ an alternating method for the determination of the mesh and the physical solution of the PDE. Beginning with an initial (uniform) mesh, we discretize the physical PDE on this fixed mesh. The numerical solution from this discretization is used to solve for a new mesh, on which the physical PDE is discretized again, and so on. This iteration is complete when both the mesh and physical solutions converge. Convergence of a single domain alternating iteration for the mesh and physical PDEs in the case of static boundary layer problems is presented in [36].

For equidistribution of solutions of time-dependent PDEs, there are two main ways to approach the discretization of the physical PDE; these are the quasi-Lagrange approach and the rezoning approach. In the quasi-Lagrange approach, the mesh points are considered to move continuously in time, forming a moving mesh PDE (MMPDE). Then, the discretized time derivatives of the physical PDE are transformed into time derivatives along the mesh. This is explained in detail in [31]. In the rezoning approach, the mesh moves intermittently in time, and so is naturally often coupled with an alternating procedure. The mesh is updated at each time step, the physical solution is interpolated onto this new mesh, and the physical PDE is discretized on the new mesh.

The coupled system of a mesh equation and physical time-dependent PDE is also generally solved in one of two ways, either simultaneously or alternately. We remark that by definition, the rezoning method can only be used with an alternating method. In this thesis, we employ a rezoning approach coupled with an alternating method.

Additionally, in the case of common PDEs such as the heat equation posed on a curved domain, one obtains different differential equation than the common form of the heat equation $u_t = u_{xx}$ in one dimension. The common differential operators such as the surface gradient and Laplace-Beltrami operators reduce to the first and second derivatives of u respectively, with additional curve-based coefficients in front of these derivatives. More detail on these operators can be found in [27] and [35].

In general, we note that our emphasis in this thesis is not on efficiency; we are not attempting to determine the overall most efficient method. There will be no direct comparison of the total computational cost of using r-refinement versus adding additional nodes to a uniform mesh. Our goal is to provide a proof of concept and show that mesh generation via r-refinement can be applied to these example problems on curves.

We also wish to solve these mesh equations and physical problems on multiple subdomains. This allows us to take advantage of parallel computing and solve the subproblems on each subdomain simultaneously. This reduces the overall computational cost of the physical solution, as well as reducing the additional cost that occurs by using r-refinement. We focus on Schwarz methods including classical and optimized Schwarz domain decomposition (DD) methods. Schwarz [48] was the first to propose an iterative (and alternating) domain decomposition technique to solve linear boundary value problems in 1870. This pioneering work has been expanded upon in multiple papers, and applied to various types of problems. An overview of popular DD methods to solve PDEs can be found in [8]. Additionally, the reference [14] provides a historical literature review of Schwarz methods.

In these Schwarz methods, the domain is partitioned into multiple subdomains, with smaller subproblems formed on each subdomain. In parallel Schwarz methods, the subproblems are independent of one another at each iteration, forming a parallel scheme. In classical Schwarz, each subdomain has a region of overlap with its adjacent subdomains, allowing us to form Dirichlet transmission conditions for the subproblems. In optimized Schwarz, the subdomains no longer need to be overlapping, resulting in Robin transmission conditions on the subproblems. In this thesis, parallelism is combined with equidistribution to solve the subproblems. This combination of equidistribution and domain decomposition on a line has been discussed in [18] and [19], with discrete analysis in [26].

Specifically, to solve time-dependent PDEs in parallel, the most natural approach is from Cai [5, 6]. In this method, we discretize the PDE in time and solve the resulting sequence of elliptic problems with domain decomposition. This is the method used in this thesis. A more recent method to solve time-dependent PDEs in parallel is called Schwarz Waveform relaxation (SWR), proposed by Gander et. al [12] and independently by Giladi [23]. In SWR, the spatial domain is decomposed and the time-dependent PDE is solved on the entire time interval on each subdomain.

The remainder of the thesis is structured as follows. Chapter 2 is an introduction to moving mesh methods on a single domain. Beginning with an explanation of the equidistribution principle, we then move on to introduce how to implement equidistribution on both an interval (line) and on a static parametric curve. We then proceed to discuss equidistribution on a moving curve (with time dependency), for which we solve the equidistribution problem at each time step. We finish the chapter with the introduction of equidistribution of a function $u(r)$ posed on a parametric curve $\mathbf{x}(r)$, for which we provide monitor functions that use both curve and function features. Chapter 3 introduces the concept of domain decomposition iterations for equidistribution. It begins with an introduction to classical and optimized Schwarz methods before specifically providing these Schwarz iterations in the case of equidistributing static curves. A brief comparison of the Schwarz algorithms is also provided. Chapter 4 introduces single domain approaches to the equidistribution of a function $u(r)$ posed on a parametric curve $\mathbf{x}(r)$, where $u(r)$ is the solution of a static boundary layer problem and is therefore unknown. We also provide theoretical results discussing the convergence of the given algorithm. We then discuss equidistribution of a time-dependent function $u(r,t)$ posed on a parametric curve $\mathbf{x}(r)$, where $u(r,t)$ is the unknown solution of a time-dependent PDE with periodic boundary conditions. In this chapter, we employ rezoning and alternating techniques that alternate between a solution of the moving mesh problem and a solution of the physical PDE. Chapter 5 introduces multi-domain iterations of the problems discussed in Chapter 4. Specifically, we provide classical Schwarz iterations that could easily be modified to form optimized Schwarz iterations. Chapter 6 concludes the thesis with a summary of the presented results, statement of contribution, and recommendations for future work.

Chapter 2

Single Domain Equidistribution

In this chapter, we introduce mesh equidistribution on a single domain. We begin with introducing the equidistribution principle on an interval, giving a mesh boundary value problem (BVP). We then extend this technique to a parametric curve $\mathbf{x}(r)$, giving a similar BVP to be solved, where the monitor function equidistributes by the curve features. We then proceed to discuss equidistribution on a time-dependent curve, using a technique where we solve a mesh BVP at each time step. Finally, we introduce equidistribution of a known function $u(r)$ posed on a parametric curve $\mathbf{x}(r)$. Here, we explore multiple monitor functions which take in a mixture of both the curve and the function features. We compare these monitor functions using interpolation errors computed on a fine grid.

2.1 Equidistribution on an Interval

Recall from Chapter 1 that on an interval $[a, b]$ broken into m sub-intervals, the EP tells us that we wish to have a set of nodes such that

$$\int_{x_0}^{x_1} M(x)dx = \int_{x_1}^{x_2} M(x)dx = \ldots = \int_{x_{m-1}}^{x_m} M(x)dx, \tag{2.1.1}$$

where $x_0 = a$ and $x_m = b$. Given an interval $x \in [a, b]$, we wish to determine an equidistributing mesh such that (2.1.1) is satisfied. We proceed on a continuous level to derive a boundary value problem whose solution determines the equidistributing mesh. We begin by introducing a computational coordinate $\xi \in [0,1]$, where

$$\xi_i = \frac{i}{m}, \qquad i = 0, \dots, m. \tag{2.1.2}$$

The points ξ_i with $\xi_0 = 0$ and $\xi_m = 1$ form a uniform mesh. The equidistributing equation (2.1.1) can be written as

$$\int_a^{x_i} M(x)dx = \frac{i}{m}\int_a^b M(x)dx, \qquad i = 0, \dots, m, \tag{2.1.3}$$

which, using the coordinate transformation, becomes

$$\int_a^{x(\xi_i)} M(x)dx = \xi_i \int_a^b M(x)dx, \qquad i = 0, \dots, m, \tag{2.1.4}$$

or on a continuous level,

$$\int_a^{x(\xi)} M(x)dx = \xi \int_a^b M(x)dx, \tag{2.1.5}$$

for all $\xi \in (0,1)$. If (2.1.5) is satisfied for a mapping $x = x(\xi)$, it is referred to as a equidistributing coordinate transformation for $M(x)$ [31].

The following theorem is taken from [31]. Theorem 2.1.1 states that in the continuous case there exists a unique equidistributing mesh of size m satisfying the equidistribution principle, provided the monitor function chosen is bounded away from zero.

Theorem 2.1.1. *For a given integer $m > 0$, there exists a unique equidistributing mesh of m points satisfying* (2.1.1) *for any strictly positive monitor function.*

The proof of Theorem 2.1.1 can be found in [31].

Many numerical methods employ (2.1.5) directly. A common method used to approximate the solution to (2.1.5) is de Boor's algorithm [7]. Convergence results for de Boor's algorithm can be found in [45] and [51]. However, we wish to implement a differential equation version of (2.1.5) in practice. This allows us to use Newton's method, a common numerical method used to solve nonlinear differential equations. This is done for multiple reasons. Pryce [45] shows that de Boor's algorithm is a fixed point iteration with local linear convergence, while Newton's method gives local quadratic convergence. Additionally, Newton's method generalizes more easily than de Boor's algorithm to higher spatial dimensions. We note that in practice the convergence of Newton's method can be difficult, as we need a sufficient number of mesh points and an initial guess that is sufficiently close to the actual solution to guarantee convergence.

To obtain a differential equation from (2.1.1), we differentiate (2.1.5) twice with respect to ξ, giving the two-point boundary value problem given by

$$\frac{d}{d\xi}\left(M(x(\xi))\frac{dx(\xi)}{d\xi}\right) = 0, \qquad x(0) = a,\;\; x(1) = b. \tag{2.1.6}$$

Solving this nonlinear boundary value problem gives the solution $x(\xi)$, which in turn gives us the coordinates of the equidistributed mesh.

2.2 Equidistribution on Curves

Given a parameterized curve $(x(r), y(r))$ for $r \in [0, b]$, we implement a coordinate transformation, as in Section 2.1, and aim to find mesh points $r_i = r(\xi_i) = r(\frac{i}{N})$ such that these points are equally distributed along the curve. We focus on a differential approach given in [39] for a parameterized curve. To determine the transformation $r(\xi)$ on a continuous level, we begin with an initial value problem given in [39], that is,

$$F(r)\frac{dr}{d\xi} = \int_0^b F(r)dr, \qquad 0 \leq r \leq b, \qquad r(0) = 0, \tag{2.2.1}$$

where $F(r) > 0$ is a chosen monitor function. Throughout this thesis, we use $F(r)$ to denote a monitor function defined on a curve and $M(x)$ to denote a monitor function defined on an interval. Differentiating this with respect to ξ gives

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) = 0, \qquad 0 \leq r \leq b, \qquad r(0) = 0,\;\; r(1) = b, \tag{2.2.2}$$

a boundary value problem analogous to (2.1.6).

We remark that it is straightforward to show that in the continuous case, the constraint $0 < r < b$ will be satisfied in the solution of (2.2.2). We can see from (2.2.1) that if $F(r) > 0$, then $\frac{dr}{d\xi} > 0$, and therefore $r(\xi)$ will be monotone. However, while the continuous solution $r(\xi)$ is guaranteed to be bounded and monotonic, the discrete solution may not satisfy these constraints during the Newton iteration. This may cause computational issues, depending on the problem. A fix to this issue is provided in Remark 5.1.1.

Common families of choices for the monitor function $F(r)$ are those corresponding to the weighted arc-length for a parametric curve, given by

$$F(r) = w(r)\sqrt{\left(\frac{dx}{dr}\right)^2 + \left(\frac{dy}{dr}\right)^2}, \tag{2.2.3}$$

and those corresponding to the weighted curvature of a parametric curve, given by

$$F(r) = w(r)\left(\left(\frac{d^2x}{dr^2}\right)^2 + \left(\frac{d^2y}{dr^2}\right)^2\right)^{\frac{1}{4}}, \tag{2.2.4}$$

where $w(r) > 0$ is a user-chosen weight function that is problem dependent. Here, we choose $w(r) = 1$; a more detailed explanation of the weight function is given in [39]. We note additionally that to ensure that $F(r) \geq \bar{F} > 0$ for all r, we can add a constant inside the square root, giving monitor functions like

$$F(r) = w(r)\left(a + \left(\frac{d^2x}{dr^2}\right)^2 + \left(\frac{d^2y}{dr^2}\right)^2\right)^{\frac{1}{4}}, \tag{2.2.5}$$

for $a > 0$. Here, $\bar{F} > 0$ for some constant "floor" $\bar{F}$ sufficiently far from zero. We note that one should also choose the weight $\omega(r)$ such that it is sufficiently large enough to ensure that $F(r)$ does not approach zero. For the n-dimensional curve $\mathbf{x}(r) = (x_1(r), x_2(r), \ldots, x_n(r))^T \in \mathbb{R}^n$, we proceed analogously to the $\mathbb{R}^2$ case to determine the grid points $r_i = r(\xi_i)$. This results in the BVP (2.2.2), with a monitor function corresponding to arc-length given by

$$F(r) = w(r)\sqrt{\mathbf{x}_r \cdot \mathbf{x}_r}, \tag{2.2.6}$$

and a monitor function corresponding to curvature given by

$$F(r) = w(r)(\mathbf{x}_{rr} \cdot \mathbf{x}_{rr})^{\frac{1}{4}}, \tag{2.2.7}$$

where $\mathbf{x}_r = (\frac{dx_1(r)}{dr}, \frac{dx_2(r)}{dr}, \ldots, \frac{dx_n(r)}{dr})^T$ and $\mathbf{x}_{rr} = (\frac{d^2x_1(r)}{dr^2}, \frac{d^2x_2(r)}{dr^2}, \ldots, \frac{d^2x_n(r)}{dr^2})^T$.

In practice, we find that using the fourth root $(\cdot)^{\frac{1}{4}}$ in the curvature-based monitor function reduces the interpolation errors when compared to using the square root

$\sqrt{(\cdot)}$. The same does not occur for arc-length-based monitor functions. Further reasoning behind choosing a fourth root instead of the square root for curvature-based expressions is discussed in Section 2.4.1.

As an example, we consider a parameterized curve given by

$$x(r) = (1+\cos(A\pi r))\cos(2\pi r), \quad y(r) = (1+\cos(A\pi r))\sin(2\pi r), \quad 0 \le r \le 1, \tag{2.2.8}$$

and aim to equidistribute the mesh points $(x(r_i), y(r_i))$ for $i = 0, \ldots, m$.

We remark that (2.2.8) is an important example as it is not a one-to-one function; i.e., it can not be represented as a function of x and does not pass the vertical line test. Additionally, the curve changes drastically depending on the choice of A. An example of how the curve changes with the choice of A is shown in Figure 2.1. We see from Figure 2.1 that as we increase A, we generally add "petals" to the curve and change the curve significantly.

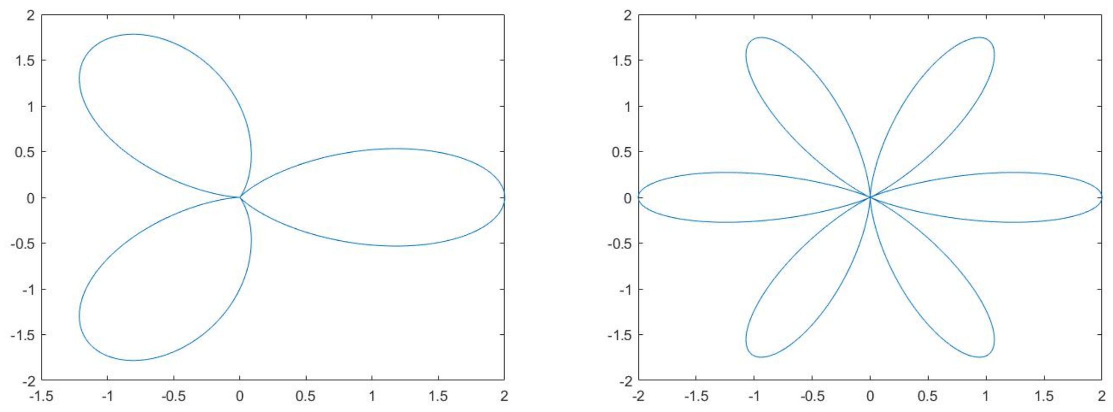


Figure 2.1: Curve (2.2.8) shown for $A = 6$ (left) and $A = 12$ (right) on a fine uniform mesh of size $m = 1000$ nodes.

For a monitor function $F(r)$ given by (2.2.6) or (2.2.7) (with $\omega(r) = 1$), we discretize the mesh BVP (2.2.2) using finite difference methods. Specifically, we discretize the left-hand side of (2.2.2) as

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) \approx \frac{1}{\Delta\xi}\left(F(r_{i+1/2})\frac{dr}{d\xi}\bigg|_{r_{i+1/2}} - F(r_{i-1/2})\frac{dr}{d\xi}\bigg|_{r_{i-1/2}}\right). \tag{2.2.9}$$

Additionally, we approximate the numerical values at the half nodes $r_{i+1/2}$ and $r_{i-1/2}$

as

$$F(r_{i+1/2})\frac{dr}{d\xi}\bigg|_{r_{i+1/2}} \approx \left(\frac{F(r_{i+1}) - F(r_i)}{2}\right)\left(\frac{r_{i+1} - r_i}{\Delta\xi}\right), \tag{2.2.10}$$

and

$$F(r_{i-1/2})\frac{dr}{d\xi}\bigg|_{r_{i-1/2}} \approx \left(\frac{F(r_{i+1}) - F(r_i)}{2}\right)\left(\frac{r_{i+1} - r_i}{\Delta\xi}\right). \tag{2.2.11}$$

Substituting (2.2.10) and (2.2.11) into (2.2.9) gives

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) \approx \frac{\left(F(r_{i+1}) + F(r_i)\right)\left(r_{i+1} - r_i\right) - \left(F(r_i) + F(r_{i-1})\right)\left(r_i - r_{i-1}\right)}{2\Delta\xi^2}, \tag{2.2.12}$$

for $i = 1, \ldots, m-1$. With the Dirichlet boundary conditions of (2.2.2) giving $r_0 = 0$ and $r_m = b$, we have formed the system of nonlinear equations. The resulting system is solved using Newton's method. This discretization gives an order of error that is approximately $O(\Delta\xi^2)$.

Results for equidistribution via arc-length and curvature are shown in Figure 2.2. As expected, equidistributing by arc-length produces equidistant nodes along the entire length L of the curve. Equidistributing by curvature produces a concentration of nodes at areas of high curvature.

We also provide Figure 2.3, which shows a closer look at the left hand side of Figure 2.2 to better view what is happening in the figure. This shows the r versus ξ plots for the interval $\xi \in [0.3, 0.8]$.

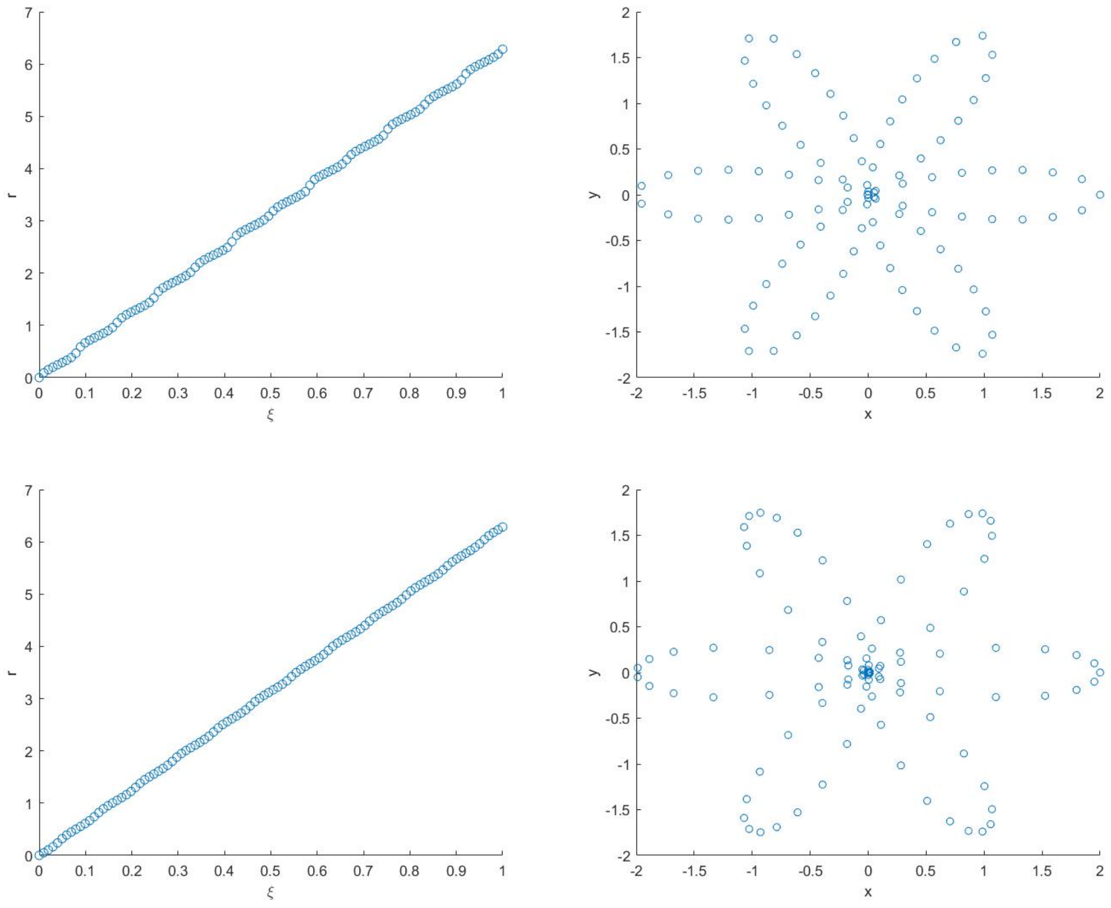


Figure 2.2: Curve (2.2.8) with $A = 12$ equidistributed by arc-length (top) and curvature (bottom) with $m = 100$ nodes. Left: a plot of r versus ξ showing the solution to (2.2.2). Right: Solution plotted at the relocated nodes.

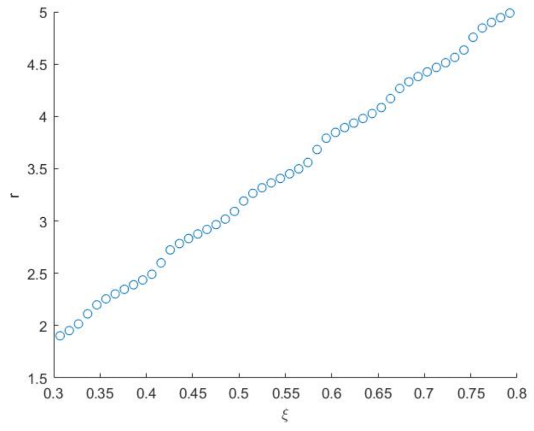


Figure 2.3: A closer look at the plots of r vs ξ for the curve (2.2.8) with $A = 12$, equidistributed with $m = 100$ nodes. Both arc-length (left) and curvature (right) based monitor functions are used.

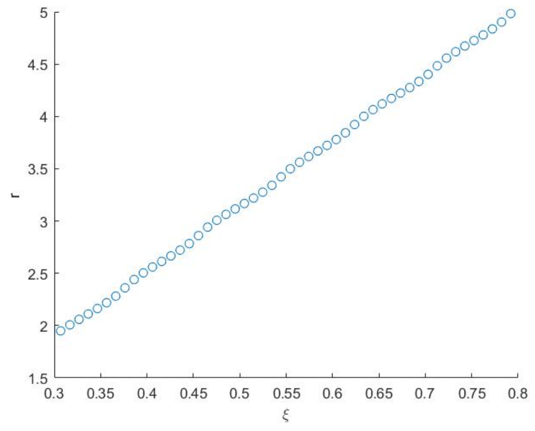
Continuing with curves that are unable to be written as a function of x, we consider the curve

$$x(r) = (1+0.5\sin(5r))\cos(r), \quad y(r) = (1+0.5\sin(5r))\sin(r), \quad 0 \le r \le 2\pi, \tag{2.2.13}$$

a curve used in [49], and the ellipse

$$x(r) = A\cos(r), \quad y(r) = B\sin(r), \quad 0 \le r \le 2\pi. \tag{2.2.14}$$

Results for equidistribution via arc-length and curvature for curves (2.2.13) and (2.2.14) are shown in Figures 2.4 and 2.5, respectively. The results are similar to that of Figure 2.2; equidistributing by arc-length produces an equal equidistribution along the arc-length L, and equidistributing by curvature produces a concentration of nodes in areas of high curvature on the curve.



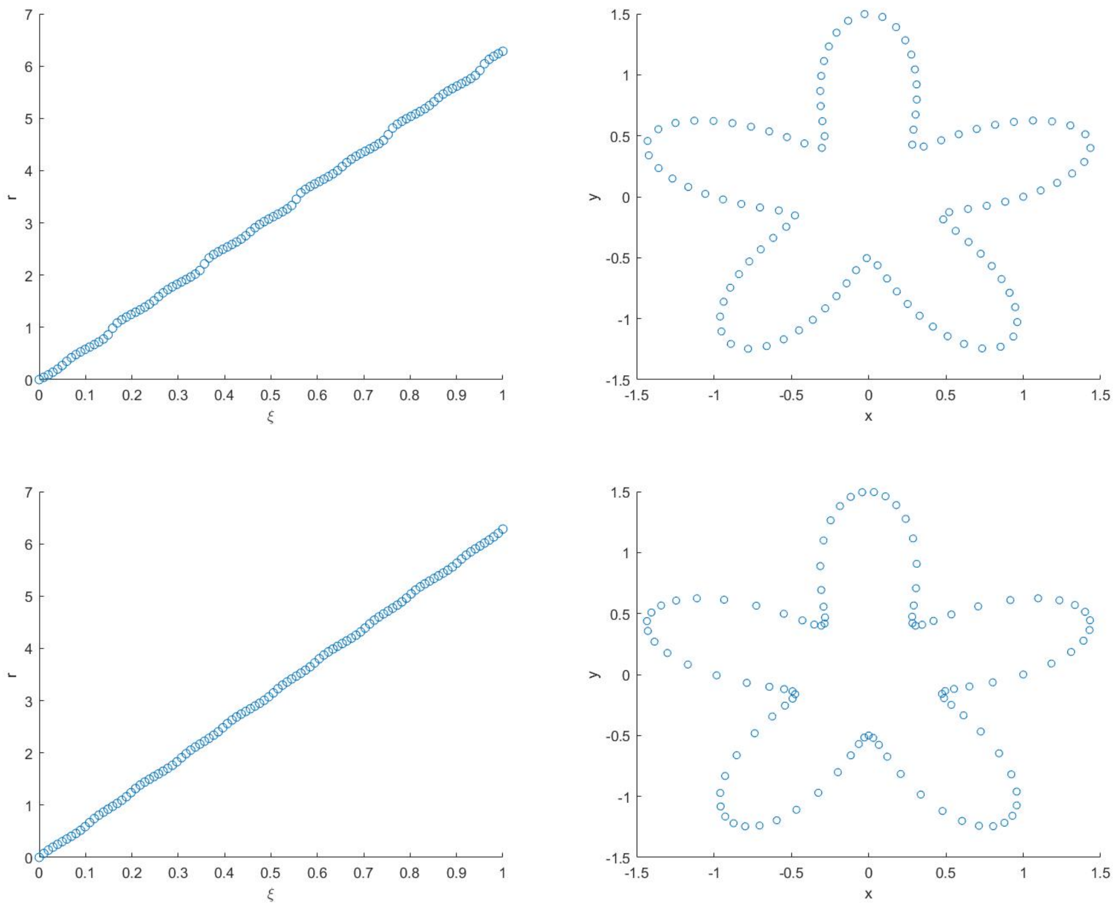


Figure 2.4: Curve (2.2.13) equidistributed by arc-length (top) and curvature (bottom) with $m = 100$ nodes. Left: a plot of r versus ξ showing the solution to (2.2.2). Right: Parameterized curve plotted at the relocated nodes.

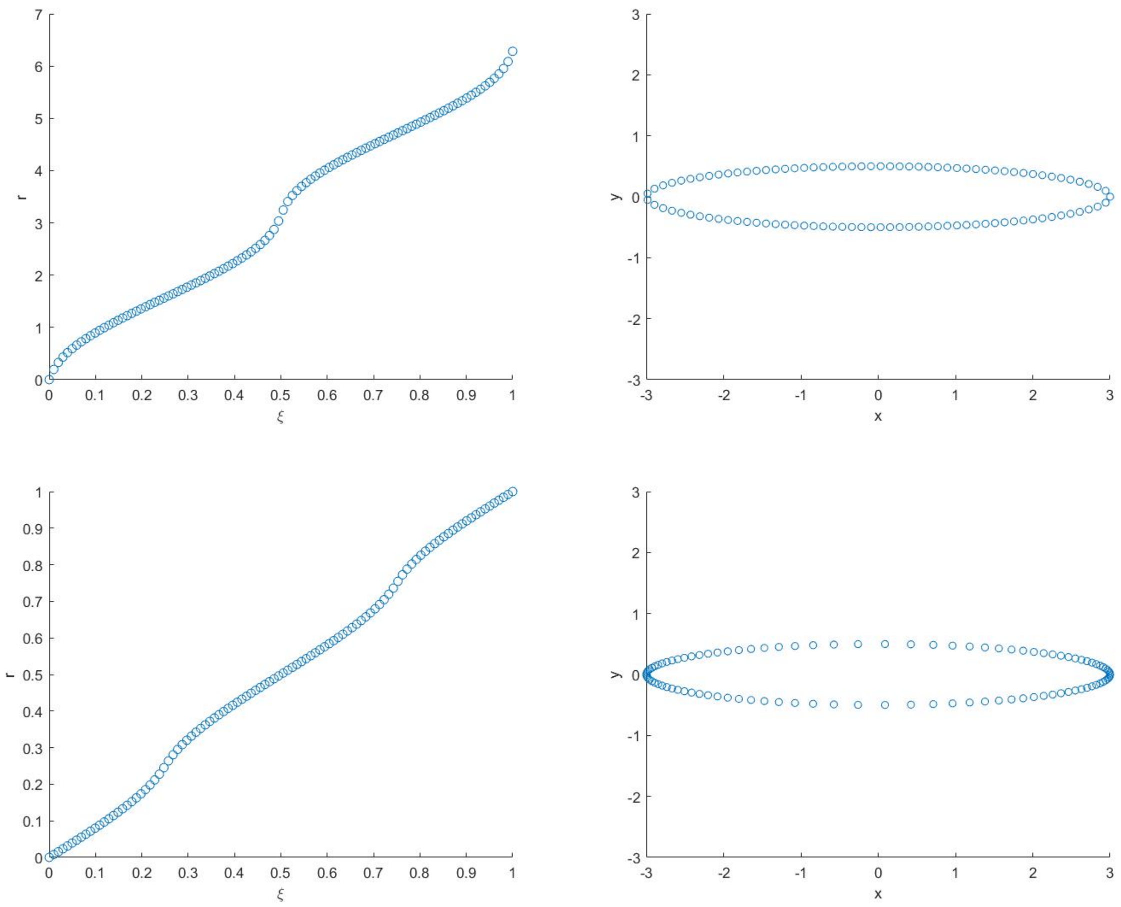


Figure 2.5: Curve (2.2.14) with $A = 3$ and $B = 0.5$ equidistributed by arc-length (top) and curvature (bottom) with $m = 100$ nodes. Left: a plot of r versus ξ showing the solution to (2.2.2). Right: Parameterized curve plotted at the relocated nodes.

To determine which monitor function produces the "best" result, we compare interpolation errors. We begin with a coarse uniform grid $r_{unif} = (\hat{r}_0, \hat{r}_1, \ldots, \hat{r}_m)^T$ and a fine uniform grid $r_{fine} = (\tilde{r}_0, \tilde{r}_1, \ldots, \tilde{r}_M)^T$, where $M = Rm$ (with integer $R > 1$) denotes the number of nodes in the fine grid. For example, if $R = 128$ and $m = 64$, then $M = 8192$. By equidistributing the coarse grid r with a given monitor function, we obtain the equidistributed (non-uniform) grid $r = (r_0, r_1, \ldots, r_m)^T$. Evaluating the curve (from the given parametric equations) gives the curve coordinates $(x(r_i), y(r_i))$ for $i = 0, \ldots, m$, and the vectors $x = x(r)$ and $y = y(r)$ are linearly interpolated onto the fine grid r_{fine} to give x_{interp} and y_{interp}. Then these interpolated values are compared to $x_{fine} = x(r_{fine})$ and $y_{fine} = y(r_{fine})$, where x_{fine} and y_{fine} are simply given by the curve evaluated on the fine uniform mesh. The difference between the equidistributed grids and the numerical fine grid solution is measured by $e = \sqrt{(x_{fine} - x_{interp})^2 + (y_{fine} - y_{interp})^2}$. In addition to the maximum norm $||e||_\infty$, we include results in the Euclidean grid norm, given by

$$||e||_2 = \left(h \sum_{i=1}^{m} e_i^2 \right)^{1/2}$$

where h is the mesh width of the fine uniform grid r_{fine}. In [37], more information regarding the motivation of this norm is given, see Appendix A. We record error norms for equidistributing both by arc-length and curvature for curves (2.2.8) with $A = 12$, (2.2.13), and (2.2.14) with $A = 3$ and $B = 0.5$ in Table 2.1. From Table 2.1, we see that equidistributing by curvature produces a smaller interpolation error when compared to the fine grid numerical solution.

Table 2.1: Interpolation errors for curves (2.2.8), (2.2.13), and (2.2.14). Here, we use $m = 100$ nodes and $M = 12800$ nodes for the fine mesh used for the interpolation error. The monitor function (2.2.6) equidistributes by arc-length, and the monitor function (2.2.7) equidistributes by curvature features.

Curve	Monitor Function	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$ (grid)
(2.2.8)	(2.2.6)	0.0659	0.0563
(2.2.8)	(2.2.7)	0.0125	0.0222
(2.2.13)	(2.2.6)	0.0247	0.0184
(2.2.13)	(2.2.7)	0.0045	0.0081
(2.2.14)	(2.2.6)	0.0165	0.0102
(2.2.14)	(2.2.7)	.000909	0.0017

We note that in [49], the authors use de Boor's algorithm with slightly different monitor functions to equidistribute along static parametric curves; these monitor functions include equidistributing with respect to both arc-length and curvature. The authors use a similar method to compare methods by measuring interpolation errors. As the authors were focused on the spectral accuracy of the equidistribution, they stated that an arc-length-based monitor function for a static curve is spectrally accurate to machine precision, while the curvature-based monitor function is not. To achieve spectral accuracy, the authors use a Fourier approximation, followed by spectral integration and spectral differentiation; see [49] for details on the method.

They also noted that qualitatively for the mean curvature flow of a curve, which is a time-dependent problem, the curvature-based monitor function gives a better approximation near high curvature regions in early time steps. However, they state that an arc-length-based monitor function is more stable in later time steps, consistently providing a smooth solution, as a high concentration of nodes in areas of high curvature may cause an insufficient amount of nodes in other areas for proper resolution.

2.3 Equidistribution on Time-Dependent Curves

So far, we have discussed equidistribution on curves and lines that are only spatially dependent. That is, there has been no dependence on time. We now focus on a time-dependent curve given by $\mathbf{x}(r,t) = (x_1(r,t), x_2(r,t), \ldots, x_N(r,t))^T \in \mathbb{R}^N$. We wish to determine a mesh $(r_1(t), r_2(t), \ldots, r_m(t))^T$ such that

$$\int_a^{r_j(t)} F(r(t),t)dr = \frac{j}{m}\int_a^b F(r(t),t)dr, \qquad j = 0, \ldots, m, \tag{2.3.1}$$

is satisfied for $t \geq 0$. It is important to note that this mesh will change and "move" as time passes, unlike the steady case. We remark that the goal of this section is to show that it is possible to use the framework considered in this thesis, with some modifications, to deal with the case of a time-dependent curve. Time-dependent curves will only be discussed in this section.

A simple example is given by the time-dependent two-dimensional curve

$$x(r) = (4\sin(t)(\tanh(r) + 2r), \quad y(r) = r^{10} + r). \tag{2.3.2}$$

We begin by evaluating the curve at multiple points in time and equidistributing the curve with the static mesh BVP (2.2.2) for each time value t. The results are given in Figure 2.6. We see from Figure 2.6 that as time changes/passes, the positions of the mesh nodes change as well.

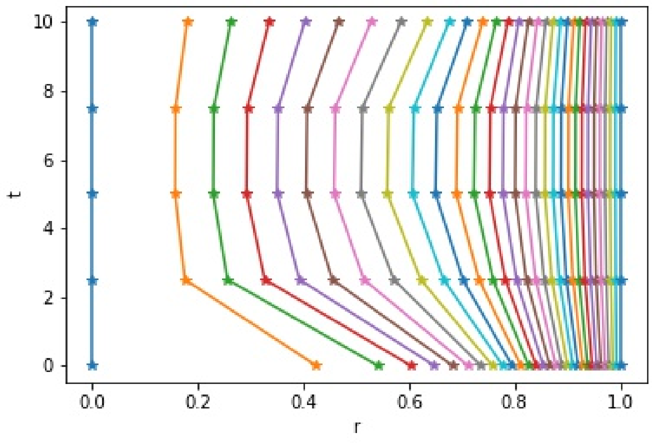


Figure 2.6: Mesh trajectory of the time-dependent curve (2.3.2) equidistributed by curvature with $m = 30$ nodes at 5 time steps between $t = 0$ and $t = 10$. The mesh r varies with time t.

To obtain the differential equation in the continuous case, we proceed as in the time-independent case. We introduce a computational coordinate $r = r(\xi, t)$ such that $\xi \in [0,1]$, $r(0,t) = a$, and $r(1,t) = b$. Differentiating (2.3.1) twice gives us the boundary value problem

$$\frac{d}{d\xi}\left(F(r(\xi,t),t)\frac{dr(\xi,t)}{d\xi}\right) = 0, \quad \forall t \geq 0. \tag{2.3.3}$$

The equation (2.3.3) is often referred to as a quasi-static equidistribution principle as it does not explicitly involve the rate of change $\frac{dr(\xi,t)}{dt}$.

On an interval, another common method to equidistribute a time-dependent function $u(r,t)$ is to form a MMPDE that explicitly involves the mesh speed $\frac{dr(\xi,t)}{dt}$, first proposed in [29]. Instead of a quasi-static equidistribution problem, we obtain a time-dependent PDE. A detailed discussion of MMPDEs can be found in [31].

2.4 Equidistribution of Functions on Static Curves

Expanding on Section 2.2, consider the equidistribution of a known function $u(r)$ posed on the curve $(x(r), y(r))$. In this section, we focus on the function

$$u(r) = \tan^{-1}\left(\frac{r - r_0}{\epsilon}\right), \tag{2.4.1}$$

for given constants r_0 and ϵ, posed on the different curves given parametrically by $(x(r), y(r))$.

We begin with (2.4.1) defined on the ellipse (2.2.14). A visual representation of the function $u(r)$ posed on $(x(r), y(r))$ is shown in Figure 2.7. This forms a 3D plot. Additionally, we provide a visual representation of the effect of ϵ on the function u. The function on the curve is plotted for multiple ϵ values in Figure 2.8. We see that as ϵ decreases, there is more need for adaptivity in the function $u(r)$ as there is a steeper slope of the function u at $r = r_0$.

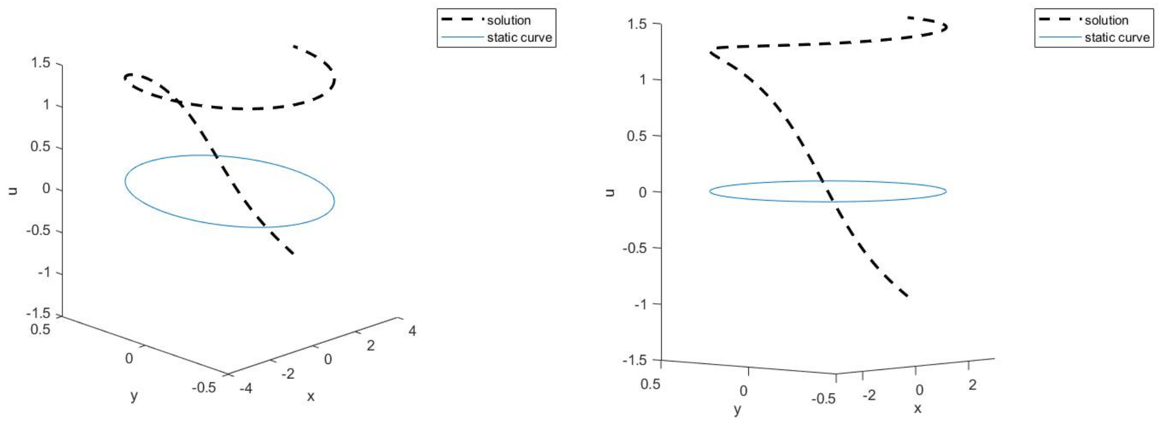


Figure 2.7: The function (2.4.1) with $r_0 = \frac{\pi}{4}$ and $\epsilon = 0.5$ posed on the ellipse (2.2.14) with $A = 3$ and $B = 0.5$. This is shown from multiple angles.

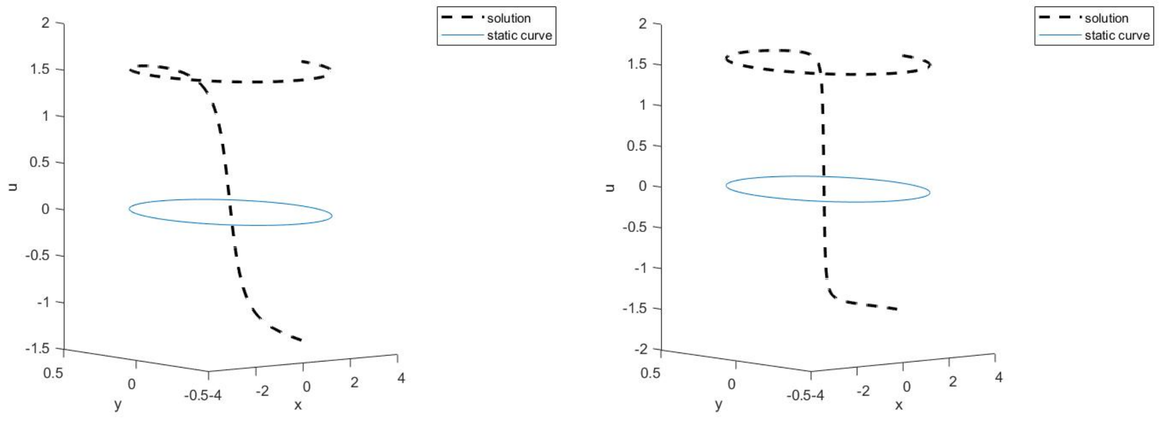


Figure 2.8: The function (2.4.1) with $r_0 = \frac{\pi}{4}$ and two values of ϵ posed on the ellipse (2.2.14) with $A = 3$ and $B = 0.5$. Plots are shown for $\epsilon = 0.1$ (left) and $\epsilon = 0.01$ (right).

In this case, we look to equidistribute the mesh nodes in a way that takes into account the features of both the static curve $\mathbf{x}(r)$ and the function $u(r)$ on $\mathbf{x}(r)$. A way to approach this problem is to introduce a monitor function $F(r)$ that includes the features of $\mathbf{x}$ and u. We introduce and test multiple monitor functions, comparing the interpolation errors in these experiments.

2.4.1 Choice of Monitor Function

We look to obtain a monitor function that contains both a "curve part" and "function part". The following monitor functions are explored:

$$F(r) = \sqrt{1 + u'(r)^2}, \tag{2.4.2}$$

$$F(r) = (1 + \kappa(r))^{\frac{1}{4}}, \tag{2.4.3}$$

$$F(r) = (1 + \kappa(r))^{\frac{1}{4}} + \sqrt{1 + u'(r)^2}, \tag{2.4.4}$$

$$F(r) = (1 + x''(r)^2 + y''(r)^2)^{\frac{1}{4}}, \tag{2.4.5}$$

$$F(r) = (1 + x''(r)^2 + y''(r)^2)^{\frac{1}{4}} + \sqrt{1 + u'(r)^2}, \tag{2.4.6}$$

and

$$F(r) = \sqrt{1 + x'(r)^2 + y'(r)^2} + \sqrt{1 + u'(r)^2}, \tag{2.4.7}$$

$$\kappa(r) = \frac{|x'(r)y''(r) - y'(r)x''(r)|}{(\sqrt{x'(r)^2 + y'(r)^2})^3}, \tag{2.4.8}$$

is the curvature of $\mathbf{x}(r) = (x(r), y(r))^T$ by definition [39]. All of the above monitor functions satisfy the condition that $F(r) \geq \bar{F} > 0$; the addition of 1 to the monitor functions ensures this "floor". In practice, when we remove this floor, i.e.,

$$F(r) = (\kappa(r))^{\frac{1}{4}}, \tag{2.4.9}$$

we often see an unstable solution due to $F(r) \approx 0$ for certain r.

As a final remark, note that in monitor functions (2.4.4), (2.4.6), and (2.4.7), we are adding two radical expressions together to form our monitor functions. We remark that when we modify these monitor functions by reforming them as the multiplication of two radical expressions such as

$$F(r) = (1 + x''(r)^2 + y''(r)^2)^{\frac{1}{4}} \sqrt{1 + u'(r)^2}, \tag{2.4.10}$$

we obtain a very similar monitor function in the sense that they have identical regions of peaks, dips, and plateaus. This is shown in Figure 2.9. We see from Figure 2.9 that the monitor functions result in steep peaks at $r = \frac{\pi}{4}$ due to the function (2.4.1) with $r_0 = \frac{\pi}{4}$, as well as peaks in the regions $r = 0$, $r = \pi$, and $r = 2\pi$ due to the curvature of the ellipse (2.2.14). We also see that multiplying the square roots in (2.4.10) produces a greater emphasis on the peak at $r = \frac{\pi}{4}$, meaning that there is a greater emphasis on the function features when compared to the monitor function (2.4.6).

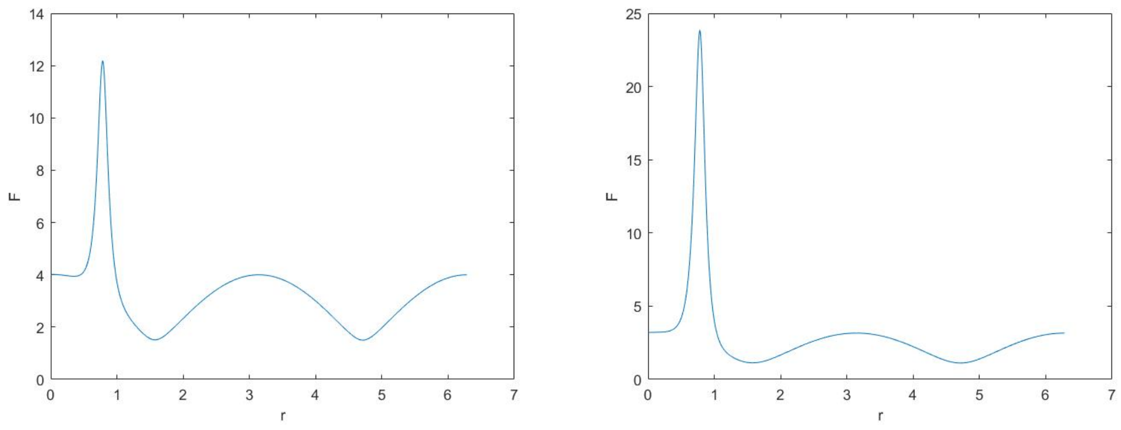


Figure 2.9: Visualization of the monitor functions produced from monitor functions (2.4.6) (left) and (2.4.10) (right). The curve $\mathbf{x}$ is given by (2.2.14) with $A = 6$ and $B = 0.5$, and the function $u(r)$ is given by (2.4.1) with $r_0 = \frac{\pi}{4}$ and $\epsilon = 0.1$.

To visualize how the values of the monitor function are influenced the curve and function features, the curve and function features are plotted in Figures 2.10 and 2.11 for the curves (2.2.13) and (2.2.14), respectively. Additionally, the bottom row of Figures 2.10 and 2.11 show the corresponding monitor functions to provide a connection to the problem features and the resulting monitor functions.

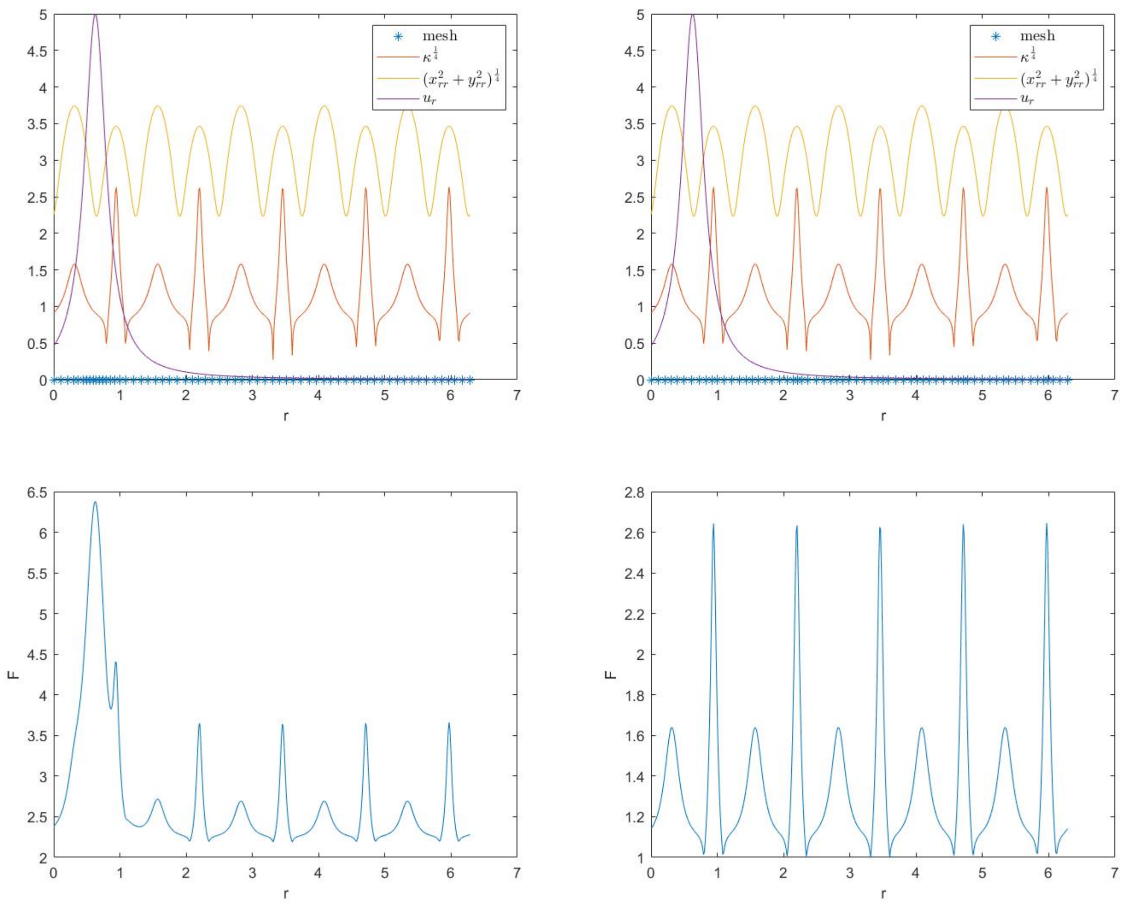


Figure 2.10: A visualization of the monitor functions (2.4.4) (left) which equidistributes by both curve and function features, and (2.4.3) (right) which only equidistributes by the curve features. The curve $\mathbf{x}$ is given by (2.2.13) and the function $u(r)$ is given by (2.4.1) with $r_0 = \frac{\pi}{5}$ and $\epsilon = 0.2$. The resulting mesh from the monitor functions is shown in $m = 64$ blue points. Top: Features of the curve $\mathbf{x}$ and the function $u(r)$. Bottom: A plot of the monitor functions $F(r)$.

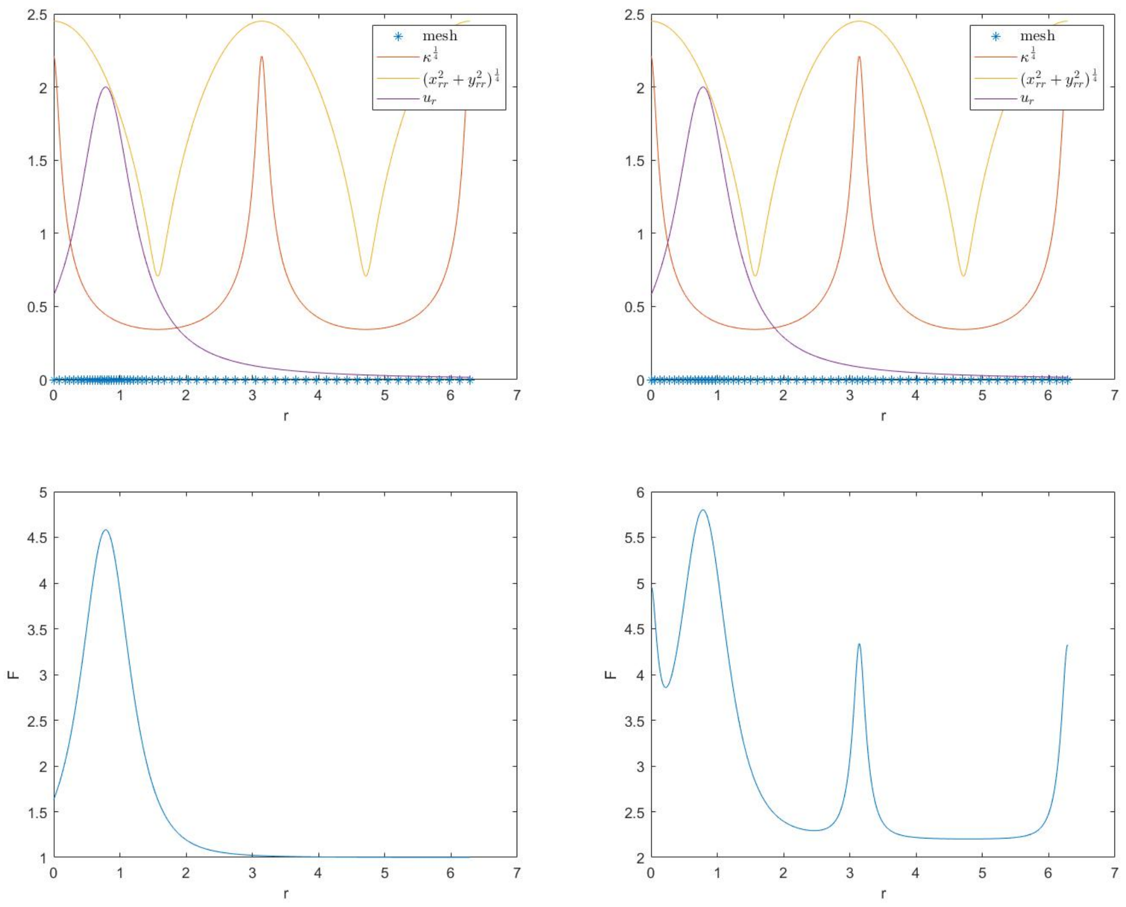


Figure 2.11: A visualization of the monitor functions (2.4.2) (left) which only equidistributes by the function features, and (2.4.4) (right) which equidistributes by both the curve and function features. The curve $\mathbf{x}$ is given by (2.2.14) with $A = 6$ and $B = 0.5$, and the function $u(r)$ given by (2.4.1) with $r_0 = \frac{\pi}{4}$ and $\epsilon = 0.5$.The resulting mesh from the monitor functions is shown in $m = 64$ blue points. Top: Features of the curve $\mathbf{x}$ and the function $u(r)$. Bottom: A plot of the monitor functions $F(r)$.

We see from Figures 2.10 and 2.11 that, as expected, the monitor function (2.4.2) only concentrates nodes where the function u is changing rapidly, (2.4.2) concentrates nodes in areas of high curvature of $\mathbf{x}$, and (2.4.4) considers both the curve and function features. Additionally, we generally note that high regions of the second derivatives $(x''(r), y''(r))$ generally correspond with high regions of curvature $\kappa(r)$, with curvature generally producing "sharper" peaks, as seen in Figures 2.10 and 2.11.

We test the above monitor functions for multiple closed curves. To quantify the accuracy of the monitor functions, we calculate the interpolation errors as we did in

Section 2.2. We begin with a coarse uniform grid $r_{unif} = (\hat{r}_0, \hat{r}_1, \ldots, \hat{r}_m)^T$ and a fine uniform grid $r_{fine} = (\tilde{r}_0, \tilde{r}_1, \ldots, \tilde{r}_M)^T$, as described in Section 2.2. Evaluating the curve from the given parametric equations gives the curve coordinates $(x(r_i), y(r_i))$ for $i = 0, \ldots, m$, and evaluating the function from the given equation $u(r)$ gives the function coordinates $u(r_i)$ for $i = 0, \ldots, m$. The vectors $x = x(r)$, $y = y(r)$, and $u = u(r)$ are linearly interpolated onto the fine grid r_{fine} to give x_{interp}, y_{interp}, and u_{interp}. Then these interpolated values are compared to $x_{fine} = x(r_{fine}), y_{fine} = y(r_{fine})$, and $u_{fine} = u(r_{fine})$, where x_{fine}, y_{fine}, and u_{fine} are simply given by the curve and function evaluated on the fine uniform mesh.

It is important to note that in order to determine the interpolation error, we can not merely determine the error of the function on the equidistributed mesh. If we merely determine the interpolation error of u on various r grids, we will not have any information of u specifically posed on the curve. In practice, since the function u is posed on the curve $\mathbf{x}$, we must find a way of computing interpolation error such that the function posed on the curve is represented. Since $u(r)$ on $\mathbf{x}(r)$ can be thought of as a 3d plot $u(x(r), y(r))$, we use a version of a 3D distance formula, $e = \sqrt{(x_{fine} - x_{interp})^2 + (y_{fine} - y_{interp})^2 + (u_{fine} - u_{interp})^2}$ in our interpolation error calculations. We remark that this is merely one way to compare the equidistributed solution to the uniform solution. There are other methods of comparison that could be considered as well such as the Hausdorff distance and the (discrete) Fréchet distance; see [32] and [9] for details.

If we consider e an overall error between the equidistributed and fine grid solutions, we can also separate the error e into a "curve" and "function" error, where the curve error is defined as

$$e_c = \sqrt{(x_{fine} - x_{interp})^2 + (y_{fine} - y_{interp})^2},$$

and the function error is defined as

$$e_u = \sqrt{(u_{fine} - u_{interp})^2}.$$

This means that while e will tell us the overall error and how well we are resolving both $\mathbf{x}$ and u, e_c and e_u will tell us how well we are specifically resolving the curve and function, respectively, and are useful to observe as well. We record error norms for the curve error e_c, function error e_u, and overall error e for the function (2.4.1).

Here, (2.4.1) is considered on the curves (2.2.13), (2.2.14), and a curve used in [49] given by

$$x(r) = r + 2\sin(r), \quad y(r) = 0.5\sin(r), \quad 0 \leq r \leq 2\pi. \tag{2.4.11}$$

in Tables 2.2, 2.4, and 2.3.

Table 2.2: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \frac{\pi}{5}$ and $\epsilon = 0.2$ posed on (2.2.13). Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh. e_c denotes the curve error, e_u denotes the function error, and e denotes the overall error.

$F(r)$	$\|\|e_c\|\|_\infty$	$\|\|e_c\|\|_2$	$\|\|e_u\|\|_\infty$	$\|\|e_u\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.028585	0.033328	0.0030512	0.0022453	0.028585	0.033403
(2.4.3)	0.017158	0.022233	0.020523	0.0096682	0.025411	0.024244
(2.4.4)	0.019318	0.024301	0.008436	0.0044218	0.019318	0.0247
(2.4.5)	0.014251	0.020998	0.02682	0.0094751	0.02937	0.023037
(2.4.6)	0.013698	0.021174	0.016867	0.0068086	0.018747	0.022242
(2.4.7)	0.025425	0.027233	0.0080643	0.0043351	0.025426	0.027576
uniform mesh	0.017302	0.022374	0.019732	0.0088396	0.022265	0.024057

Table 2.3: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \frac{\pi}{4}$ and $\epsilon = 0.2$ posed on (2.2.14) with $A = 6$ and $B = 0.5$. Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh. e_c denotes the curve error, e_u denotes the function error, and e denotes the overall error.

$F(r)$	$\|\|e_c\|\|_\infty$	$\|\|e_c\|\|_2$	$\|\|e_u\|\|_\infty$	$\|\|e_u\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.012458	0.014604	0.0030787	0.0023088	0.012458	0.014785
(2.4.3)	0.007319	0.0094711	0.021995	0.0098047	0.022597	0.013632
(2.4.4)	0.009584	0.010844	0.007195	0.0042168	0.009584	0.011635
(2.4.5)	0.011079	0.011183	0.022131	0.011159	0.022794	0.015798
(2.4.6)	0.0065466	0.0089005	0.010574	0.0058171	0.010868	0.010633
(2.4.7)	0.035088	0.02691	0.010604	0.0057552	0.035088	0.027518
uniform	0.0074585	0.0096877	0.019416	0.0088579	0.02031	0.013127

Table 2.4: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \frac{\pi}{5}$ and $\epsilon = 0.2$ posed on (2.4.11). Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh. e_c denotes the curve error, e_u denotes the function error, and e denotes the overall error.

$F(r)$	$\|\|e_c\|\|_\infty$	$\|\|e_c\|\|_2$	$\|\|e_u\|\|_\infty$	$\|\|e_u\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.0042662	0.0050763	0.0030512	0.0022453	0.0042662	0.0055507
(2.4.3)	0.0029989	0.0034238	0.023732	0.010722	0.023779	0.011256
(2.4.4)	0.0041667	0.0041189	0.0083988	0.0049419	0.0084166	0.0064333
(2.4.5)	0.0059212	0.004654	0.054162	0.026694	0.054484	0.027096
(2.4.6)	0.0021576	0.0032119	0.0074302	0.0041411	0.0074457	0.0052407
(2.4.7)	0.0054863	0.0055855	0.0051601	0.0028131	0.0055024	0.0062539
uniform	0.0025619	0.0033171	0.019732	0.0088396	0.019809	0.0094415

Tables 2.2, 2.3, and 2.4 tell us valuable things about the equidistribution in a quantitative sense. We see that how well the equidistribution performs is problem dependent; however, each example produces at least one monitor function for which the equidistribution gives more accuracy than a uniform mesh in all norms.

From Table 2.2, we see that for the curve (2.2.13), the monitor function (2.4.6) produces the lowest overall interpolation error $||e||$ in both the maximum and Euclidean norm. The curve error $||e_c||$ is minimized by the monitor function (2.4.6) in the maximum norm, and (2.4.5) in the Euclidean norm. The function error $||e_u||$ is minimized by the monitor function (2.4.2) in both the maximum and Euclidean norms. We recall that (2.4.6) equidistributes by both curve and function features, (2.4.5) equidistributes by only curve features, and (2.4.2) equidistributes by only function features.

From Table 2.3, we see that for the curve (2.2.14), the monitor function (2.4.4) produces the lowest error in the maximum norm and the monitor function (2.4.6) produces the lowest overall interpolation error in the Euclidean norm. The curve error $||e_c||$ is minimized by the monitor function (2.4.6) in both the maximum and Euclidean norms. The function error $||e_u||$ is minimized by the monitor function (2.4.2) in both the maximum and Euclidean norms. We recall that (2.4.4) equidistributes by both curve and function features.

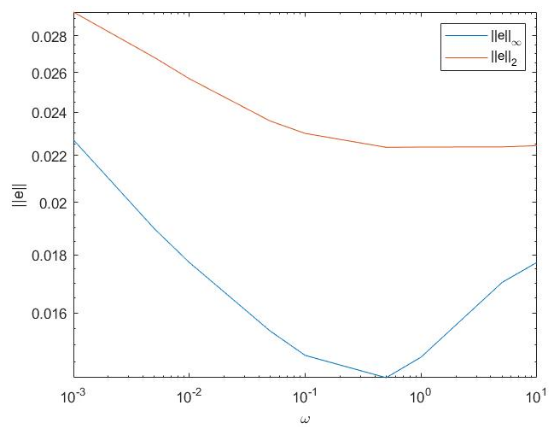
From Table 2.4, we see that for the curve (2.4.11), the monitor function (2.4.2) produces the lowest error in the maximum norm and the monitor function (2.4.6) produces the lowest overall interpolation error in the Euclidean norm. The curve error $||e_c||$ is minimized by the monitor function (2.4.6) in both the maximum and Euclidean norms. The function error $||e_u||$ is minimized by the monitor function (2.4.2) in both the maximum and Euclidean norms.

This means that for these examples, the overall best performing monitor functions are those that take in both the curve and function features. Here, "best" is defined as producing the lowest overall errors $||e||_2$ and e_∞. This is an important discovery, as it provides numerical evidence of our earlier comments. While the errors are not reduced for the majority of monitor functions, we do obtain lower overall errors with certain monitor functions. Although the errors are not reduced significantly in Tables 2.2, 2.3, and 2.4, it is important to note that our goal is not to optimize the choice of monitor function. We have chosen a selection of monitor functions that appear sensible but have made no attempt to determine the overall optimal monitor function. We have provided monitor function selections and shown that even among this small selection with no attempt of optimization, there exist monitor functions that produce a reduction in interpolation error when compared to a uniform mesh.

We note additionally that weights $\omega, \omega_u > 0$ other than $\omega = \omega_u = 1$ can be placed at any point in a monitor function; this can be used to put emphasis on curve or function features. As an example, we experiment with weights in front of (2.4.6); this gives the monitor function

$$F(r) = \sqrt{1 + \omega(x''(r)^2 + y''(r)^2)} + \sqrt{1 + \omega_u u'(r)^2}. \tag{2.4.12}$$

As an exercise, we fix both the monitor function and the curve (2.2.13) and experiment with the weights. Results are given in Figure 2.12.



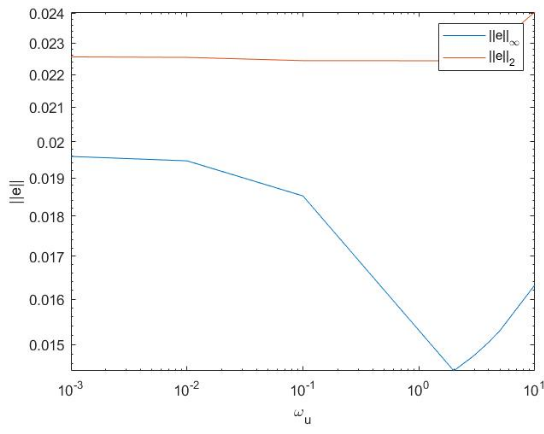


Figure 2.12: Figures comparing values of ω and ω_u in the monitor function (2.4.12) for the equidistribution of (2.4.1) with $r_0 = \frac{\pi}{5}$ and $\epsilon = 0.2$ posed on the curve (2.2.13). Here, we used $m = 64$ nodes and $M = 8192$ nodes for the fine mesh. Interpolation errors are shown in the maximum and Euclidean norms for fixed $\omega_u = 1$ and varied values of ω (left), and fixed $\omega = 1$ and varied values of ω_u (right).

From Figure 2.12, in this example, we get better results when more weight is placed on the function features. However, we see that there is an optimal weight combination to minimize interpolation error in this example, given by $\omega = 0.5, \omega_u = 1$ in the maximum and Euclidean norms. However, this is likely due to the fact that $u(r)$ changes more drastically with r in certain regions and "needs" equidistribution more than the curve $\mathbf{x}$. We conclude that the weights will be problem dependent.

A Monitor Function Based on Interpolation Error

If we are determining the "success" of a monitor function by the resulting interpolation error, then we should consider a monitor function that is designed with interpolation in mind. As previously stated, we use $M(x)$ to denote a monitor function on an interval and $F(r)$ to denote a monitor function on a curve. The authors of [31] explored this in Section 2.5 of their book. The authors determined that an interpolation-based monitor function given by

$$M(x) = \left(1 + \frac{1}{\alpha}(u''(x))^2\right)^{1/5}, \tag{2.4.13}$$

with weight given by

$$\alpha(x) = \left(\frac{1}{b-a}\int_a^b (u''(x))^{2/5}\, dx\right)^5,$$

minimizes the linear interpolation error in the L^2 norm. The authors also remarked that on an interval, the monitor function given by

$$M(x) = \left(1 + u_{xx}^2\right)^{1/4}, \tag{2.4.14}$$

is motivated by the fact that the second derivative of a function generally has ties to interpolation error. In practice, a monitor function that uses the second derivative of u is related to a monitor function which reduces the interpolation error. The authors give error bounds in the L^2 norm for these monitor functions of

$$||e||_2 \leq C\left(\int_a^b |u''|^{\frac{2}{5}} dx\right)^{\frac{5}{2}}, \tag{2.4.15}$$

for the optimal monitor function (2.4.13), and

$$||e||_2 \leq C\left(\int_a^b \frac{|u''|^2}{1+|u''|^2} dx\right)^{\frac{1}{2}}\left(\int_a^b (1+|u''|^2)^{\frac{1}{4}} dx\right)^2, \tag{2.4.16}$$

for the curvature-based monitor function (2.4.14), where C is constant and $x \in (a, b)$.

On the curve $(x(r), y(r))$ with the function $u(r)$, a few monitor functions of this type would look like

$$F(r) = (1 + u''(r)^2)^{\frac{1}{4}}, \tag{2.4.17}$$

$$F(r) = (1 + \kappa(r))^{\frac{1}{4}} + (1 + u''(r)^2)^{\frac{1}{4}}, \tag{2.4.18}$$

and

$$F(r) = (1 + x''(r)^2 + y''(r)^2)^{\frac{1}{4}} + (1 + u''(r)^2)^{\frac{1}{4}}. \tag{2.4.19}$$

Results comparing the interpolation errors for these monitor functions are provided in Tables 2.5, 2.6, and 2.7. For these monitor functions (2.4.17), (2.4.18), and (2.4.19), we compare them to (2.4.2), (2.4.4), and (2.4.6), their respective analogous monitor functions that only differ in using $u'(r)$ instead of $u''(r)$. We see that in Tables 2.5, 2.6, and 2.7, using $u''(r)$ in the monitor functions (2.4.17), (2.4.18), and (2.4.19) is generally not producing a noticeably lower interpolation error in the maximum and Euclidean norms when compared to (2.4.2), (2.4.4), and (2.4.6).

We note that while (2.4.14) is designed to minimize interpolation error and we have extended this principle to form (2.4.5), (2.4.18), and (2.4.19), it is still to be explored theoretically if these new monitor functions will minimize interpolation error. As mentioned, we are not seeing this for this example; this could be due to a number of reasons as we are extending (2.4.14) to the curve case. We remark that in Section (4.2) for a different example problem, the monitor function (2.4.19) produces the minimum interpolation error out of all selected monitor functions. We emphasize that the overall "best" monitor function will be problem dependent and that theoretical work is necessary to identify optimal monitor functions for equidistribution on curves.

Table 2.5: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \frac{\pi}{5}$ and $\epsilon = 0.2$ posed on (2.2.13). Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh.

$F(r)$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.02821	0.033064
(2.4.17)	0.030334	0.034779
(2.4.4)	0.019509	0.024888
(2.4.18)	0.020221	0.025137
(2.4.6)	0.015142	0.022955
(2.4.19)	0.015516	0.02303
uniform mesh	0.022265	0.024057

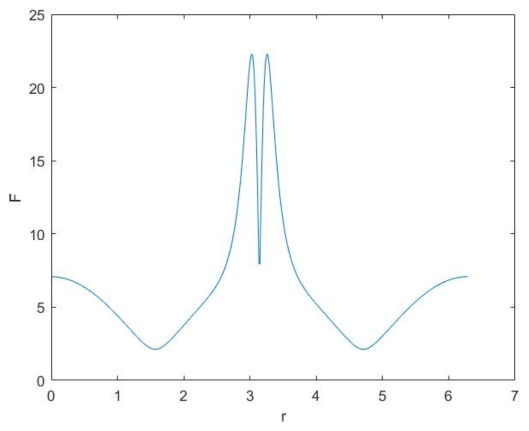
Table 2.6: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \pi$ and $\epsilon = 0.2$ posed on (2.2.14) with $A = 6$ and $B = 0.5$. Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh. A smoothing parameter $p = 1$ was used in the monitor functions.

$F(r)$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.012275	0.013457
(2.4.17)	0.013356	0.013864
(2.4.4)	0.0091175	0.011154
(2.4.18)	0.0095018	0.010935
(2.4.6)	0.0065292	0.0096542
(2.4.19)	0.0067589	0.0095954
uniform mesh	0.020673	0.013117

Table 2.7: Interpolation errors for the equidistribution of (2.4.1) with $r_0 = \pi$ and $\epsilon = 0.2$ posed on (2.4.11). Here, we use $m = 64$ nodes and $M = 8192$ nodes for the fine mesh.

$F(r)$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.2)	0.0042023	0.0058965
(2.4.17)	0.0045818	0.0059711
(2.4.4)	0.0064238	0.0056514
(2.4.18)	0.0062468	0.0053926
(2.4.6)	0.0066976	0.0054884
(2.4.19)	0.006473	0.0051595
uniform mesh	0.019296	0.0094452

In practice, for problems such as (2.4.1), $u''(r)$ produces a very nonsmooth monitor function, with multiple "peaks" very close to each other. This is shown in Figure 2.13. This issue can generally be fixed with sufficient smoothing. This is discussed in further detail in Chapter 4.



(a) (2.4.4)

Figure 2.13: Monitor function (2.4.19), with the curve given by (2.2.14) with $A = 6$ and $B = 0.5$, and function $u(r)$ given by (2.4.1) with $r_0 = \pi$ and $\epsilon = 0.2$.

Chapter 3

Domain Decomposition

Domain decomposition (DD) is a divide and conquer method used to aid in the numerical solution of PDEs by partitioning the domain into multiple smaller subdomains and solving subproblems on each subdomain. To complete these subproblems, transmission conditions are introduced to complete the definition of the PDE on each subdomain.

In this chapter, we begin with an introduction of classical and optimized Schwarz DD iterations for general linear problems $\mathcal{L}u = f$ in Sections 3.1.1 and 3.1.2. Classical Schwarz involves Dirichlet transmission conditions on each subdomain problem, and requires an overlap between subdomains. Optimized Schwarz involves Robin transmission conditions on each subdomain problem, and requires only a common boundary between the subdomains. This will be explained further throughout the chapter. Next, we specifically state the classical and optimized Schwarz iterations used to solve the curve equidistribution problem given by (2.2.2). Numerical evidence is provided throughout to show that both iterations converge to the single domain solution of (2.2.2). In Section 3.2.1, we provide analysis closely linked to the analysis of [18] to show convergence of the parallel classical Schwarz iteration. In Section 3.2.2, we provide the discretization and implementation of the modified Robin transmission conditions and provide analysis closely linked to the analysis of [18] to show convergence of the parallel optimized Schwarz iteration. We also provide a brief analysis of the parameter p, a user chosen constant in the optimized Schwarz iteration. This analysis includes a description of the upper bound on p and a numerical experiment to compare various values of p. We finish the chapter with a direct comparison of the classical and optimized Schwarz iterations.

3.1 General DD Methods

3.1.1 Classical Schwarz

The original domain decomposition scheme to solve boundary value problems was proposed by Schwarz [48]; the author proposed an alternating/sequential (non-parallel) domain decomposition method to prove the Dirichlet principle. This domain decomposition method was later expanded upon with parallel variants. In this thesis, we will focus on parallel domain decomposition, which allows us to invoke parallel computing. The first extension of the alternating Schwarz method to a parallel Schwarz method was proposed by Lions [38]. Parallel computing has garnered a large amount of interest as of late, with growing access to parallel computers and multiple cores.

Consider a domain $\Omega = [0, b]$ decomposed into two subdomains Ω_1 and Ω_2. Given initial guesses $u_1^{(0)}$ and $u_2^{(0)}$, the general alternating Schwarz iteration for a (time-independent) problem, is given by the following: for $n = 0, 1, 2, \ldots$, solve

$$\begin{aligned} \mathcal{L}u_1^{(n+1)} &= f, \quad x \in \Omega_1, \quad \mathcal{L}u_2^{(n+1)} = f, \quad x \in \Omega_2, \\ u_1^{(n+1)} &= u_2^{(n)}, \quad x \in \Gamma_1, \quad u_2^{(n+1)} = u_1^{(n)}, \quad x \in \Gamma_2, \end{aligned} \tag{3.1.1}$$

where $\Gamma_1 = \partial\Omega_1 \cap \Omega_2$ and $\Gamma_2 = \partial\Omega_2 \cap \Omega_1$. Here, $\partial\Omega_i$ denotes the boundary of Ω_i. An important note is that classical Schwarz requires overlapping subdomains; i.e., $\Omega_i \cap \Omega_{i+1} \neq 0$ for subdomain Ω_i. In (3.1.1), the subproblems are independent of each other at each iteration, allowing the iteration to be carried out in parallel.

Extending the iteration to multiple subdomains, the algorithm is given by the following: for $n = 0, 1, 2, \ldots$, solve

$$\begin{aligned} \mathcal{L}u_i^{(n+1)} &= f, \qquad x \in \Omega_i, \\ u_i^{(n+1)} &= u_j^{(n)}, \qquad x \in \Gamma_{ij}, \end{aligned} \tag{3.1.2}$$

for $i = 1, \ldots, S$ and j such that $\Gamma_{ij} = \partial\Omega_i \cap \Omega_j$ is non-empty. In one-dimensional space such that $\Omega = [0, b]$, this means that Ω_j is adjacent to Ω_i, that is, $j = i - 1$ and $j = i + 1$ for $i = 2, \ldots, S - 1$. Here, the domain Ω is decomposed into S overlapping subdomains $\Omega_1, \Omega_2, \ldots, \Omega_S$ such that $\Omega = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_S$.

The general alternating (non-parallel) Schwarz iteration is given by the following:

for $n = 0, 1, 2, \ldots$, solve

$$\begin{aligned} \mathcal{L}u_i^{(n+1)} &= f, \qquad x \in \Omega_i, \\ u_i^{(n+1)} &= u_j^{(n+1)}, \qquad x \in \Gamma_{ij}, \end{aligned} \tag{3.1.3}$$

for $i = 1, \ldots, S$. For all iterations discussed in Section 3.1.1, we assume that at most two subdomains are contained in any part of the domain. In other words, in one dimension on an interval $[a, b]$ with m subdomains, this gives

$$\Omega_i = [\alpha_i, \beta_i], \quad \text{for} \quad i = 1, \ldots, S,$$

with $\alpha_i \leq \beta_{i-1} \leq \alpha_{i+1}$. Here, $\alpha_1 = a$ and $\beta_S = b$.

These alternating and parallel Schwarz iterations are classical Schwarz methods. Often, alternating and parallel Schwarz iterations are referred to as multiplicative and additive Schwarz methods, respectively. The classical Schwarz methods involve Dirichlet boundary conditions on each subdomain, requiring a multi-node overlap between the subdomains. Without this overlap, the Dirichlet boundary conditions will not necessarily impose smoothness at the boundaries of the domains. In this case the iteration would generally not produce a smooth solution, and therefore would not converge to the single domain solution.

3.1.2 Optimized Schwarz on two Subdomains

Optimized Schwarz methods can be traced to Lions [38]; he proposed Robin transmission conditions to remove the overlap requirement of classical Schwarz. These Robin boundary conditions (often called "mixed" boundary conditions) introduce a constant p, which provides a weight for the Dirichlet part of the boundary condition.

As in classical Schwarz, we partition the domain into S subdomains $\Omega_1, \Omega_2, \ldots, \Omega_S$. However, in optimized Schwarz for $S = 2$ subdomains, the overlap can be reduced to merely a common boundary such that $\Gamma_1 = \Gamma_2 = \Gamma$. With 2 subdomains on an interval $[a, b]$, this would give $\Omega_1 = [a, \alpha]$ and $\Omega_2 = [\alpha, b]$ where $\Gamma = \alpha$. Given initial guesses $u_1^{(0)}$ and $u_2^{(0)}$, the general parallel optimized Schwarz iteration to solve the linear problem $\mathcal{L}u = f, x \in \Omega$ on two subdomains is then given by: for $n = 0, 1, \ldots$, solve

$$\mathcal{L}u_1^{(n)} = f,\ \ x \in \Omega_1, \qquad \mathcal{L}u_2^{(n)} = f,\ \ x \in \Omega_2, \tag{3.1.4}$$

$$(\partial_{n_1} + p_1)u_1^{(n)} = (\partial_{n_1} + p_1)u_2^{(n-1)},\ \ x \in \Gamma_1,\ (\partial_{n_2} + p_2)u_2^{(n)} = (\partial_{n_2} + p_2)u_1^{(n-1)},\ \ x \in \Gamma_2, \tag{3.1.5}$$

such that each subdomain problem has Robin boundary conditions. Here, ∂_{n_i} denotes the partial derivative along the outward normal to the boundary of Ω_i, and p_i are chosen constants used to accelerate the convergence. Further discussion on the choice of boundary operators can be found in [13].

While the rate of convergence of the optimized Schwarz iterations depends on the constants p_1 and p_2, optimized Schwarz methods generally converge faster than classical Schwarz methods without requiring extra computational cost. These optimized Schwarz methods have been proven to converge for elliptic problems with subdomains that share only a common boundary [14].

3.2 DD for Equidistribution on Static Curves

3.2.1 Classical Schwarz

Recalling the single domain boundary-value problem (2.2.2) which equidistributes the mesh points along the curve, we propose a convergent multiplicative Schwarz iteration for the solution of (2.2.2) on each subdomain.

Suppose the computational domain ξ is decomposed into $S = 2$ subdomains such that $\Omega_1 = [0, \beta]$, $\Omega_2 = [\alpha, 1]$ with $\alpha < \beta$.

The proposed iteration is as follows: for $n = 0, 1, \ldots$, solve

$$\frac{d}{d\xi}\left(F(r_1^{(n)})\frac{dr_1^{(n)}}{d\xi}\right) = 0,\ \ \xi \in \Omega_1, \qquad \frac{d}{d\xi}\left(F(r_2^{(n)})\frac{dr_2^{(n)}}{d\xi}\right) = 0,\ \ \xi \in \Omega_2, \tag{3.2.1}$$
$$r_1^{(n)}(0) = 0, \qquad r_2^{(n)}(\alpha) = r_1^{(n)}(\alpha),$$
$$r_1^{(n)}(\beta) = r_2^{(n-1)}(\beta), \qquad r_2^{(n)}(1) = b.$$

Parallel computing allows us to separate the problem into multiple subproblems, each of which can be solved on an independent processor. The parallel iteration is as follows: for $n = 0, 1, \ldots$, solve

$$\frac{d}{d\xi}\left(F(r_1^{(n)})\frac{dr_1^{(n)}}{d\xi}\right) = 0, \;\; \xi \in \Omega_1, \qquad \frac{d}{d\xi}\left(F(r_2^{(n)})\frac{dr_2^{(n)}}{d\xi}\right) = 0, \;\; \xi \in \Omega_2, \tag{3.2.2}$$
$$r_1^{(n)}(0) = 0, \qquad r_2^{(n)}(\alpha) = r_1^{(n-1)}(\alpha),$$
$$r_1^{(n)}(\beta) = r_2^{(n-1)}(\beta), \qquad r_2^{(n)}(1) = b.$$

The iterations (3.2.1) and (3.2.2) are stopped when the subsequent approximations agree within a tolerance ϵ, as in

$$\max(||r_1^{(n)} - r_1^{(n-1)}||_\infty, ||r_2^{(n)} - r_2^{(n-1)}||_\infty) < \epsilon. \tag{3.2.3}$$

As a simple example, we return to the curve given by (2.2.14) with $A = 3$ and $B = 0.5$. Both parallel and multiplicative Schwarz DD results on $S = 2$ subdomains are shown in Figure 3.1. For classical Schwarz iterations, we define the overlap O as the number of nodes contained in the overlap between the subdomains, inclusive of the endpoints. For example, subdomains given by $\Omega_1 = (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6)^T$ and $\Omega_2 = (0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)^T$ correspond to $O = 3$ points of overlap between Ω_1 and Ω_2.

We see from Figure 3.1 that both iterations converge to the single domain solution (in black). We also provide a plot showing the iterates at $n = 1, 5$, and 20 iterations in Figure 3.2. We see from Figure 3.2 that the parallel iteration converges fairly quickly.

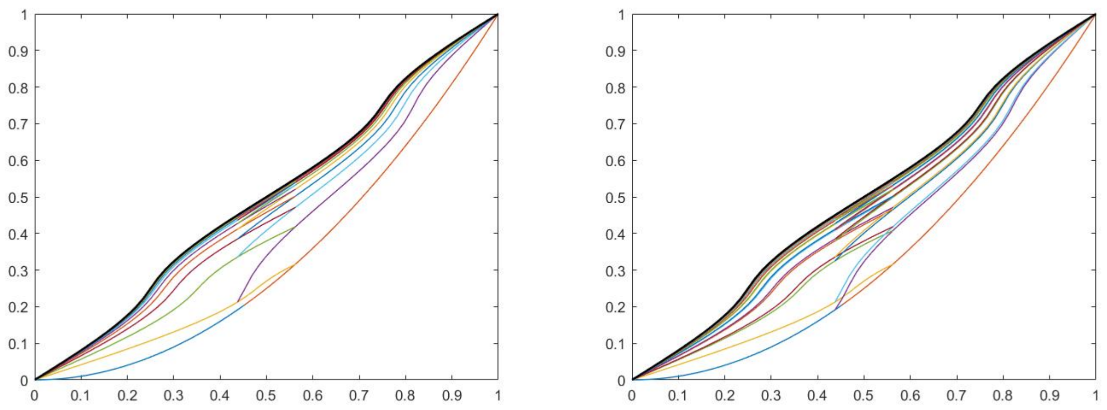
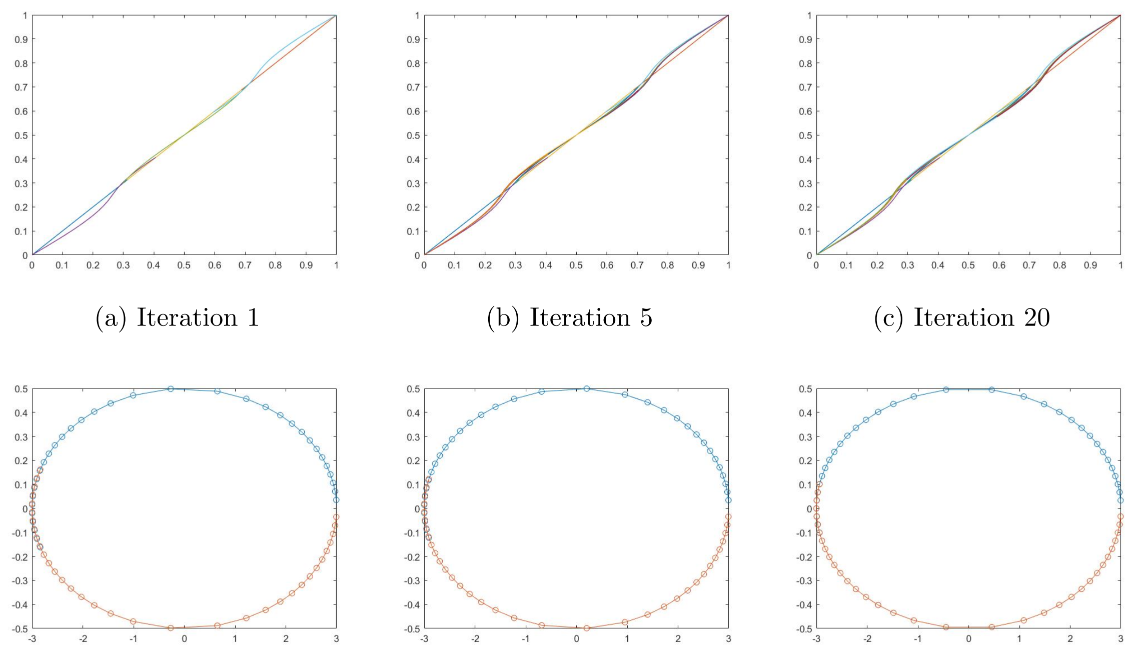


Figure 3.1: A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ from the alternating classical Schwarz algorithm (3.2.1) (left) and the parallel classical Schwarz algorithm (3.2.2) (right), where the final solution is denoted by the thick black line. Here, $m = 64$ nodes with $O = 8$ points of overlap are equidistributed by the curvature-based monitor function (2.2.7) on the curve (2.2.14), and the initial guess is given by $r_0 = \xi^2$. The alternating iteration converged in $n = 34$ iterations and the parallel iteration converged in $n = 65$ iterations; in each case the tolerance was chosen as $\epsilon = 10^{-8}$.



(a) Iteration 1 (b) Iteration 5 (c) Iteration 20

(d) Iteration 1 (e) Iteration 5 (f) Iteration 20

Figure 3.2: A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ (top) and the curve plotted at the equidistributed nodes $\left(x(r_1^{(n)}), y(r_1^{(n)})\right)$ and $\left(x(r_2^{(n)}), y(r_2^{(n)})\right)$ (bottom) from the parallel DD iteration (3.2.2), where the DD iterations are plotted after $n = 1, 5$, and 10 iterations. Here, $m = 64$ nodes with $O = 8$ points of overlap are equidistributed by the curvature-based monitor function (2.2.7) on the curve (2.2.14), and the initial guess is given by $r_0 = \xi$.

It is fairly straightforward to extend the iteration to S overlapping subdomains. For $i = 1, \dots, S$, we have $\Omega_i = [\alpha_i, \beta_i]$ with $\alpha_1 = 0$ and $\beta_S = 1$. The multiplicative Schwarz iteration is as follows: For $n = 0, 1, \dots$, solve

$$\frac{d}{d\xi}\left(F(r_i^{(n)})\frac{dr_i^{(n)}}{d\xi}\right) = 0, \qquad \xi \in \Omega_i, \tag{3.2.4}$$
$$r_i^{(n)}(\alpha_i) = r_{i-1}^{(n)}(\alpha_i), \qquad r_i^{(n)}(\beta_i) = r_{i+1}^{(n-1)}(\beta_i),$$

for subdomains $i = 1, \ldots, S$. The parallel variant is as follows: for $n = 0, 1, \ldots$, solve

$$\frac{d}{d\xi}\left(F(r_i^{(n)})\frac{dr_i^{(n)}}{d\xi}\right) = 0, \qquad \xi \in \Omega_i, \tag{3.2.5}$$
$$r_i^{(n)}(\alpha_i) = r_{i-1}^{(n-1)}(\alpha_i), \qquad r_i^{(n)}(\beta_i) = r_{i+1}^{(n-1)}(\beta_i),$$

DD results for $S = 3$ subdomains are shown in Figure 3.3. These results are similar to the results on two subdomains in Figure 3.1. We see that the main difference between the alternating and parallel results is that the left Dirichlet boundary value on $\Omega_i, i > 1$ is taken from the current iteration in the alternating scheme, and the previous iteration in the parallel scheme.

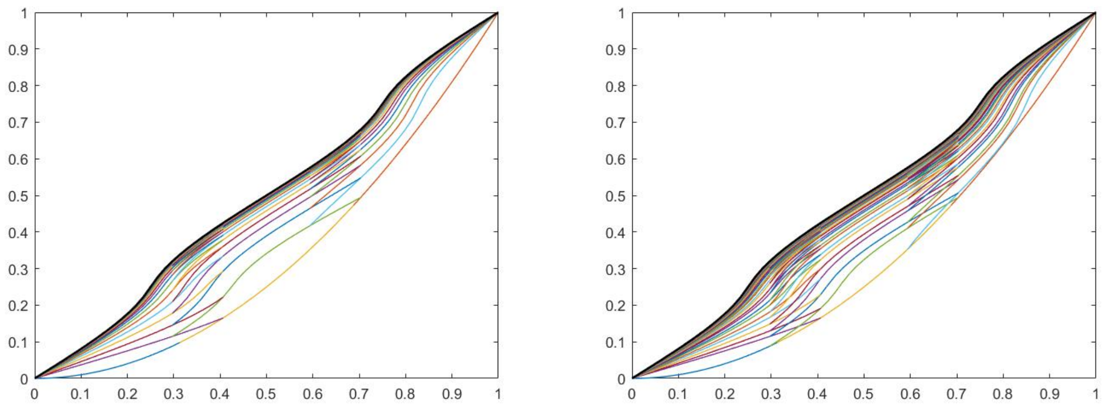


Figure 3.3: A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ from the alternating classical Schwarz algorithm (3.2.4) (left) and the parallel classical Schwarz algorithm (3.2.5) (right), on $S = 3$ subdomains, where the final solution is denoted by the thick black line. Here, $m = 64$ nodes with $O = 8$ points of overlap are equidistributed by the curvature-based monitor function (2.2.7) on the curve (2.2.14), and the initial guess is given by $r_0 = \xi$. The alternating iteration converged in $n = 53$ iterations and the parallel iteration converged in $n = 99$ iterations; in each case the tolerance was chosen as $\epsilon = 10^{-8}$.

We have also provided plots of DD error showing convergence to the single domain solution for both the multiplicative and parallel variants in Figure 3.4. We see from Figure 3.4 that the DD solution converges to the single domain solution as expected.

It is expected that as overlap increases, the number of iterations needed for convergence decreases. These results are shown in Figure 3.5. We note that while increasing the overlap decreases the total number of iterations of the algorithm, this also results in larger problem to be solved on each subdomain. Generally, there is an optimal overlap value that will minimize total computational time.

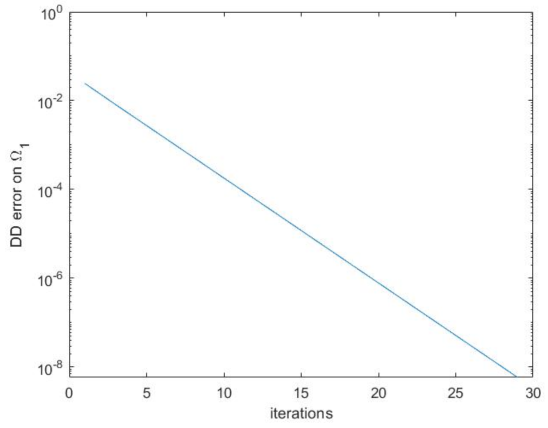


Figure 3.4: A plot of DD error versus iterations (semi-log scale) for the parallel DD iteration (3.2.2) on $S = 2$ subdomains for the ellipse (2.2.14), where the curvature-based monitor function (2.2.7) is used. Here, $m = 64$ with $O = 8$ points of overlap and initial guess is given by $r_0 = \xi$. DD error refers to the error between the DD solution and the single domain solution.

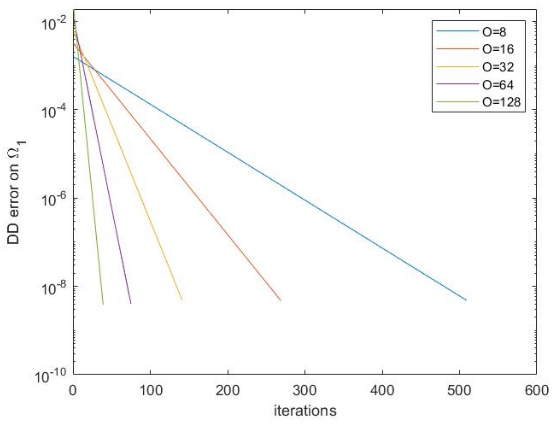


Figure 3.5: A plot of DD error versus iterations (semi-log scale) for various overlap values O on $S = 2$ subdomains. The DD error is shown for the parallel DD iteration (3.2.2) for the ellipse (2.2.14), where the curvature-based monitor function (2.2.7) is used. Here, $m = 640$ nodes are equidistributed by curvature and the initial guess is given by $r_0 = \xi$. DD error refers to the error between the DD solution and the single domain solution.

Analysis on two subdomains

We remark that the equivalent iterations to (3.2.2) and (3.2.5) for equidistribution on a 1D line were proven to converge by Gander and Haynes [18]. Using a slight adaptation of the analysis of [18], we now proceed to show convergence of the parallel classical iteration (3.2.2) for equidistribution on a curve, beginning with the well-posedness of the single domain problem on an arbitrary domain $[a, b]$.

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) = 0, \qquad r(a) = \gamma_a, \;\; r(b) = \gamma_b. \tag{3.2.6}$$

Lemma 3.2.1. *If the monitor function $F(r)$ is differentiable and bounded such that there exist constants c_0 and c_1 such that $0 < c_0 \leq F(r) \leq c_1 < \infty$ for $r \in [0, 1]$, then the BVP* (3.2.6) *has a unique solution given implicitly by*

$$\int_{\gamma_a}^{r(\xi)} F(\tilde{r})d\tilde{r} = \frac{\xi - a}{b - a}\int_{\gamma_a}^{\gamma_b} F(\tilde{r})d\tilde{r}, \quad \xi \in (a, b). \tag{3.2.7}$$

Proof. The differential equation and boundary condition at $\xi = a$ is satisfied by $\int_{\gamma_a}^{r(\xi)} F(\tilde{r})d\tilde{r} = c(\xi - a)$, where c is chosen to satisfy the Dirichlet boundary condition at the other endpoint $\xi = b$. This results in $c = \frac{1}{b-a}\int_{\gamma_a}^{\gamma_b} F(\tilde{r})d\tilde{r}$, giving (3.2.7).

We now show that the solution $r(\xi)$ from (3.2.7) exists and is unique. Here, $r(\xi)$ is the solution θ of

$$G(\theta) = \frac{\xi - a}{b - a}\int_{\gamma_a}^{\gamma_b} F(\tilde{r})d\tilde{r}, \tag{3.2.8}$$

where we define $G(\theta) \equiv \int_{\gamma_a}^{\theta} F(\tilde{r})d\tilde{r}$. Clearly from the assumptions of Lemma 3.2.1, G is continuous. Additionally, G is uniformly monotonic such that $\frac{dG}{d\theta} = F(\theta) \geq c_0 > 0$. Hence, by the implicit function theorem, there exists a unique C^1 solution to (3.2.8) (and therefore (3.2.7)). $\square$

We proceed as in [18], using Lemma 3.2.1 to construct implicit solutions on the subdomains.

Lemma 3.2.2. *Under the assumptions of Lemma 3.2.1, the subdomain solutions are given implicitly by*

$$\int_0^{r_1^{(n)}(\xi)} F(\tilde{r})d\tilde{r} = \frac{\xi}{\beta}\int_0^{r_2^{(n-1)}(\beta)} F(\tilde{r})d\tilde{r} \tag{3.2.9}$$

on Ω_1, *and*

$$\int_{r_2^{(n)}(\xi)}^{1} F(\tilde{r})d\tilde{r} = \frac{1-\xi}{1-\alpha}\int_{r_2^{(n-1)}(\alpha)}^{1} F(\tilde{r})d\tilde{r} \tag{3.2.10}$$

on Ω_2.

Proof. The proof follows directly from the general solution (3.2.7) given in Lemma 3.2.1 and is approached in an identical manner. □

Using these implicit subdomain solutions, we provide a convergence estimate for the nonlinear parallel Schwarz iteration (3.2.2). The proof is analogous to [18].

Theorem 3.2.1. *Under the assumptions of Lemma 3.2.1, the iteration* (3.2.2) *converges for any initial guess values* $r_1^{(0)}(\alpha)$ *and* $r_2^{(0)}(\beta)$. *Additionally, we have the linear convergence estimate*

$$||r - r_1^{(2n+1)}||_\infty \le \rho^n \frac{c_1}{c_0}|r(\beta) - r_2^{(0)}(\beta)|, ||r - r_2^{(2n+1)}||_\infty \le \rho^n \frac{c_1}{c_0}|r(\alpha) - r_1^{(0)}(\alpha)|, \tag{3.2.11}$$

with contraction factor $\rho \equiv \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha} < 1$.

Proof. Using Lemma 3.2.1, the sequence $r_1^{(n)}(\alpha)$ satisfies

$$\begin{aligned}\int_0^{r_1^{(n)}(\alpha)} F(\tilde{r})d\tilde{r} = \frac{\alpha}{\beta}\int_0^{r_2^{(n-1)}(\beta)} F(\tilde{r})d\tilde{r} = \frac{\alpha}{\beta}\left(\int_0^1 F(\tilde{r})d\tilde{r} - \int_{r_2^{(n-1)}(\beta)}^1 F(\tilde{r})d\tilde{r}\right) \\ \qquad (3.2.12) \\ = \frac{\alpha}{\beta}\left(\int_0^1 F(\tilde{r})d\tilde{r} - \frac{1-\beta}{1-\alpha}\int_{r_1^{(n-2)}(\beta)}^1 F(\tilde{r})d\tilde{r}\right) \\ = \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}\int_0^{r_1^{(n-2)}(\alpha)} F(\tilde{r})d\tilde{r} + \frac{\alpha}{\beta}\frac{\beta-\alpha}{1-\alpha}\int_0^1 F(\tilde{r})d\tilde{r},\end{aligned}$$

where $\int_{r_2^{(n-1)}(\beta)}^1 F(\tilde{r})d\tilde{r} = \frac{1-\beta}{1-\alpha}\int_{r_1^{(n-2)}(\beta)}^1 F(\tilde{r})d\tilde{r}$ follows from (3.2.10) evaluated at $\xi = \beta$ at iteration $n-1$.

Defining $K_1^{(n)} = \int_0^{r_1^{(n)}(\alpha)} F(\tilde{r})d\tilde{r}$ and defining $I = \int_0^1 F(\tilde{r})d\tilde{r}$, relation (3.2.12) yields the linear fixed point iteration

$$K_1^{(n)} = \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}K_1^{(n-2)} + \frac{\alpha}{\beta}\frac{\beta-\alpha}{1-\alpha}I. \tag{3.2.13}$$

Since $\rho < 1$ in this iteration, the iteration converges to a limit K_1^* given by the equation

$$K_1^* = \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}K_1^* + \frac{\alpha}{\beta}\frac{\beta-\alpha}{1-\alpha}I,$$

or

$$K_1^*\left(1 - \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}\right) = \frac{\alpha}{\beta}\frac{\beta-\alpha}{1-\alpha}I,$$

giving

$$\begin{aligned} K_1^* &= \frac{\frac{\alpha}{\beta}\frac{\beta-\alpha}{1-\alpha}}{1-\frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}}I, \\ &= \frac{\alpha(\beta-\alpha)}{\beta(1-\alpha)}\frac{\beta(1-\alpha)}{(\beta-\alpha)}I, \\ &= \alpha I. \end{aligned} \tag{3.2.14}$$

Similarly on Ω_2, we have the fixed point iteration

$$K_2^{(n)} = \frac{\alpha}{\beta}\frac{1-\beta}{1-\alpha}K_2^{(n-2)} + \frac{\beta-\alpha}{1-\alpha}I, \tag{3.2.15}$$

where $K_2^{(n)} = \int_0^{r_2^{(n)}(\beta)} F(\tilde{r})d\tilde{r}$, giving the limit

$$\begin{aligned} K_2^* &= \frac{\frac{\beta-\alpha}{1-\alpha}}{1-\frac{\alpha(1-\beta)}{\beta(1-\alpha)}}I, \\ &= \beta I. \end{aligned} \tag{3.2.16}$$

So we have obtained $\lim_{n\to\infty}\int_0^{r_1^{(n)}(\alpha)} F(\tilde{r})d\tilde{r} = \alpha\int_0^1 F(\tilde{r})d\tilde{r}$ and $\lim_{n\to\infty}\int_0^{r_2^{(n)}(\beta)} F(\tilde{r})d\tilde{r} = \beta\int_0^1 F(\tilde{r})d\tilde{r}$. In order to prove convergence to the correct limit, we note the single domain solution r also satisfies $\alpha\int_0^1 F(\tilde{r})d\tilde{r} = \int_0^{r(\alpha)} F(\tilde{r})d\tilde{r}$ and $\beta\int_0^1 F(\tilde{r})d\tilde{r} = \int_0^{r(\beta)} F(\tilde{r})d\tilde{r}$. We now have convergence to the correct limits on Ω_1 and Ω_2, giving

$$\lim_{n\to\infty}\int_0^{r_1^{(n)}(\alpha)} F(\tilde{r})d\tilde{r} = \int_0^{r(\alpha)} F(\tilde{r})d\tilde{r},$$

and

$$\lim_{n\to\infty}\int_0^{r_2^{(n)}(\beta)} F(\tilde{r})d\tilde{r} = \int_0^{r(\beta)} F(\tilde{r})d\tilde{r}.$$

We continue to prove the convergence estimate in the L^∞ norm given in Theorem 3.2.1. Subtracting equation (3.2.13) from (3.2.14) and proceeding by induction, we have

$$\int_{r_1^{(2n)}(\alpha)}^{r(\alpha)} F(\tilde{r})d\tilde{r} = \rho^n \int_{r_1^{(0)}(\alpha)}^{r(\alpha)} F(\tilde{r})d\tilde{r}, \tag{3.2.17}$$

on Ω_1. Similarly, subtracting (3.2.15) from (3.2.16) and proceeding by induction, we have

$$\int_{r_2^{(2n)}(\beta)}^{r(\beta)} F(\tilde{r})d\tilde{r} = \rho^n \int_{r_2^{(0)}(\beta)}^{r(\beta)} F(\tilde{r})d\tilde{r} \tag{3.2.18}$$

on Ω_2.

Subtracting (3.2.9) and (3.2.10) from the equivalent expressions for the single domain solution $r(\xi)$, we obtain

$$\int_{r_1^{(2n+1)}(\xi)}^{r(\xi)} F(\tilde{r})d\tilde{r} = \frac{\xi}{\beta}\int_{r_2^{(2n)}(\beta)}^{r(\beta)} F(\tilde{r})d\tilde{r},$$

and

$$\int_{r_2^{(2n+1)}(\xi)}^{r(\xi)} F(\tilde{r})d\tilde{r} = \frac{1-\xi}{1-\alpha}\int_{r_1^{(2n)}(\alpha)}^{r(\alpha)} F(\tilde{r})d\tilde{r}.$$

Substituting the equalities from (3.2.17) and (3.2.18), we have

$$\int_{r_2^{(2n+1)}(\xi)}^{r(\xi)} F(\tilde{r})d\tilde{r} = \frac{1-\xi}{1-\alpha}\rho^n \int_{r_1^{0}(\alpha)}^{r(\alpha)} F(\tilde{r})d\tilde{r}. \tag{3.2.19}$$

and

$$\int_{r_1^{(2n+1)}(\xi)}^{r(\xi)} F(\tilde{r})d\tilde{r} = \frac{\xi}{\beta}\rho^n \int_{r_2^{(0)}(\beta)}^{r(\beta)} F(\tilde{r})d\tilde{r}. \tag{3.2.20}$$

For any $a, b \in \mathbb{R}$, from the boundedness of F, we obtain

$$c_0|a-b| \leq |\int_a^b F(\tilde{r})d\tilde{r}| \leq c_1|a-b|. \tag{3.2.21}$$

Taking the modulus of (3.2.20) and (3.2.19) and applying the bound from (3.2.21), we obtain

$$|r(\xi) - r_1^{(2n+1)}(\xi)| \leq \frac{\xi}{\beta}\rho^n \frac{c_1}{c_0}|r(\beta) - r_2^{(0)}(\beta)|, \tag{3.2.22}$$

on Ω_1, and

$$|r(\xi) - r_2^{(2n+1)}(\xi)| \leq \frac{1-\xi}{1-\alpha}\rho^n \frac{c_1}{c_0}|r(\alpha) - r_1^{(0)}(\alpha)|, \tag{3.2.23}$$

on Ω_2.

Taking the supremums of (3.2.23) and (3.2.22) gives the convergence estimates (3.2.11). □

3.2.2 Optimized Schwarz on two Subdomains

The optimized Schwarz iteration for the equidistribution problem (2.2.2) has a slight variation to the boundary operators provided in Section 3.1.2. Since we know the form of the BVP that we are wishing to solve is given by

$$\frac{d}{d\xi}\left(F(r)\frac{dr}{d\xi}\right) = 0,$$

we can slightly modify the Robin boundary conditions to ease the analysis and numerical computation. A more detailed explanation of this is given in [18].

The iteration on two subdomains $\Omega_1 = [0, \beta]$ and $\Omega_2 = [\alpha, b]$ is as follows: for $n = 0, 1, \ldots$, solve

$$\begin{aligned} \frac{d}{d\xi}\left(F(r_1^{(n)})\frac{dr_1^{(n)}}{d\xi}\right) &= 0, \xi \in \Omega_1, & \frac{d}{d\xi}\left(F(r_2^{(n)})\frac{dr_2^{(n)}}{d\xi}\right) &= 0, \xi \in \Omega_2, \\ r_1^{(n)}(0) &= 0, & \mathcal{B}_2(r_2^{(n)}(\alpha)) &= \mathcal{B}_2(r_1^{(n-1)}(\alpha)), \\ \mathcal{B}_1(r_1^{(n)}(\beta)) &= \mathcal{B}_1(r_2^{(n-1)}(\beta)), & r_2^{(n)}(1) &= b, \end{aligned} \tag{3.2.24}$$

where the transmission operators are given by $\mathcal{B}_1 \equiv F(\cdot)\frac{\partial(\cdot)}{\partial\xi} + pI(\cdot)$ and $\mathcal{B}_2 \equiv F(\cdot)\frac{\partial(\cdot)}{\partial\xi} - pI(\cdot)$. As discussed in Section 3.1.2, the Robin boundary conditions allow us to choose subdomains with only a common boundary such that $\alpha = \beta$.

As an example, we look to equidistribute the ellipse (2.2.14) with $A = 3$ and $B = 0.5$. A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ showing the iterates from the optimized Schwarz scheme (3.2.24) are shown in Figure 3.6. It is important to note that these iterates are different than classical Schwarz iterates as there is no guaranteed continuity between the solution at iteration n on subdomain 1 and the solution at iteration n on subdomain 2, due to the lack of overlap between the subdomains.

We also provide an example of the iterates for the ellipse (2.2.14) with $A = 3$ and

$B = 0.5$ in Figure 3.7. We note that again, before convergence, the iterates on Ω_1 and Ω_2 are not guaranteed to be continuous.

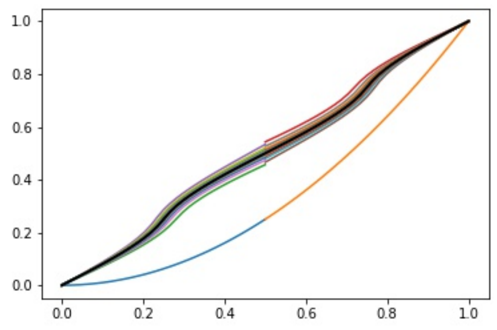
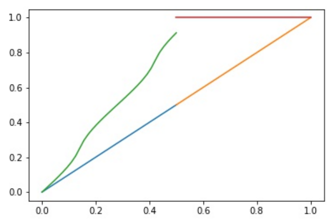
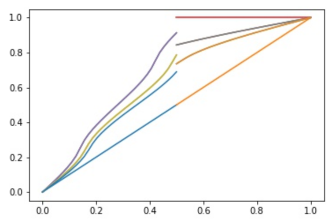


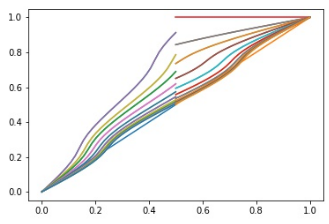
Figure 3.6: A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ from the optimized Schwarz algorithm (3.2.24), where the final solution is denoted by the thick black line. Here, $m = 64$ nodes are equidistributed by the curvature-based monitor function (2.2.7) on the curve (2.2.14), and the initial guess is given by $r_0 = \xi^2$. The parallel iteration converged in $n = 61$ iterations.



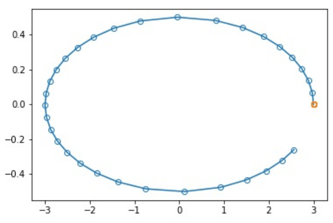
(a) Iteration 1



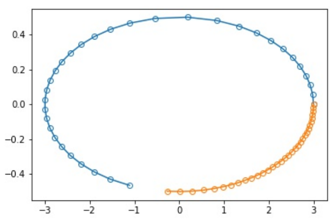
(b) Iteration 5



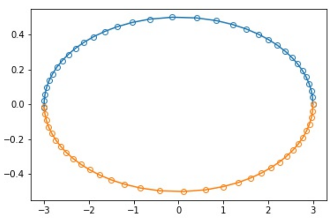
(c) Iteration 20



(d) Iteration 1



(e) Iteration 5



(f) Iteration 20

Figure 3.7: A plot of $r_1^{(n)}$ and $r_2^{(n)}$ versus ξ (top) and the curve plotted at the equidistributed nodes $\left(x(r_1^{(n)}), y(r_1^{(n)})\right)$ and $\left(x(r_2^{(n)}), y(r_2^{(n)})\right)$ (bottom) from the optimized Schwarz iteration (3.2.24), where the DD iterations are plotted after $n = 1, 5,$ and 10 iterations. Here, the constant $p = 10$ for optimized Schwarz and $m = 64$ nodes are equidistributed by the curvature-based monitor function (2.2.7) on the curve (2.2.14), and the initial guess is given by $r_0 = \xi$.

We note that some of these subdomain BVPs are difficult to solve numerically in practice. If numerically necessary to solve the equations, a scaling parameter was introduced in the monitor function. Equidistributing by curvature features, the monitor function becomes

$$F(r) = \sqrt[4]{\frac{1}{a}\left(1 + \mathbf{x}_{rr} \cdot \mathbf{x}_{rr}\right)}, \tag{3.2.25}$$

for large $a > 1$, and we use this solution to eventually solve the problem with the monitor function corresponding to $a = 1$. This is a form of continuation.

Discretization

While the discretization of the classical Schwarz iteration is fairly straightforward due to Dirichlet boundary conditions, the optimized Schwarz discretization is less intuitive. We provide a brief description of the discretization of (3.2.24). To discretize the continuous iteration (3.2.24), we discretize the domains $\Omega_1 = [0, \alpha]$ and $\Omega_2 = [\alpha, 1]$ into subdomains with mesh sizes of m_1 and m_2 nodes, respectively.

To implement the transmission conditions, we must first discretize B_r on Ω_2 and B_l on Ω_1, where B_r and B_l are given by the right hand side of the boundary conditions and can be calculated from the previous iteration such that

$$B_r = F(r_2^{(n-1)}(\beta))\frac{dr_2^{(n-1)}(\beta)}{d\xi} + pr_2^{(n-1)}(\beta),$$

and

$$B_l = F(r_1^{(n-1)}(\alpha))\frac{dr_1^{(n-1)}(\alpha)}{d\xi} - pr_1^{(n-1)}(\alpha).$$

If we choose only a common boundary between r_1 and r_2 such that $\beta = \alpha$, then as the subdomains do not overlap, we cannot use a centered difference method to compute B_l and B_r at $x = \alpha$. Instead, we proceed recursively to approximate B_l and B_r. We note that higher order one-sided finite difference derivative approximations could be performed here instead, but this recursive method typically gives better results when the subdomains are not overlapping [28].

Let $g_1^{(n)}$ and $g_2^{(n)}$ denote the nth approximation to B_r and B_l respectively, which are initially taken to be $g_1^{(1)} = g_2^{(1)} = 0$. Here, we suppress the argument α such that $r_1^{(n)}$ refers to $r_1^{(n)}(\alpha)$ for convenience. From (3.2.24) with $n = 1$, we have

$$F(r_1^{(1)})\frac{dr_1^{(1)}}{d\xi} + pr_1^{(1)} = g_1^{(1)}, \tag{3.2.26}$$

and

$$F(r_2^{(1)})\frac{dr_2^{(1)}}{d\xi} + pr_2^{(1)} = g_2^{(1)}, \tag{3.2.27}$$

Similarly for iteration $n = 2$, we have

$$\begin{aligned} g_1^{(2)} &= F(r_1^{(2)})\frac{dr_1^{(2)}}{d\xi} + pr_1^{(2)} \\ &= F(r_2^{(1)})\frac{dr_2^{(1)}}{d\xi} + pr_2^{(1)} \\ &= pr_2^{(1)} + g_2^{(1)} + pr_2^{(1)}, \\ &= 2pr_2^{(1)}, \end{aligned}$$

and

$$\begin{aligned} g_2^{(2)} &= F(r_2^{(2)})\frac{dr_2^{(2)}}{d\xi} + pr_2^{(2)} \\ &= F(r_1^{(1)})\frac{dr_1^{(1)}}{d\xi} + pr_1^{(1)} \\ &= g_1^{(1)} - pr_1^{(1)} - pr_1^{(1)}, \\ &= -2pr_1^{(1)}. \end{aligned}$$

This eventually gives the relations $B_r \equiv F(r_1^{(n)})\frac{dr_1^{(n)}}{d\xi} + pr_1^{(n)} = 2pr_2^{(n-1)} + g_2^{(n-1)}$ and $B_l \equiv F(r_2^{(n)})\frac{dr_2^{(n)}}{d\xi} + pr_2^{(n)} = -2pr_1^{(n-1)} + g_1^{(n-1)}$.

Once we have determined B_l and B_r from the previous iteration, we proceed to discretize the iteration.

We begin with Ω_1. On $\Omega_1 = [0, \alpha]$, we must discretize the right boundary condition

$$F(r_1^{(n)}(\beta))\frac{dr_1^{(n)}(\alpha)}{d\xi} + pr_1^{(n)}(\alpha) = B_r. \tag{3.2.28}$$

We proceed by introducing and subsequently eliminating a ghost point, a common method used when discretizing Neumann boundary conditions. Let $r_1 = r$, $\xi_1 = \xi$ and $m_1 = m$ for convenience such $x_m = \alpha$. Using centered differencing with $h = \Delta\xi$, we know that (3.2.28) is approximated by

$$F(r_m)\left(\frac{r_{m+1} - r_{m-1}}{2h}\right) - pr_m = B_r, \tag{3.2.29}$$

giving

$$r_{m+1} = r_{m-1} + \frac{2h}{F(r_m)}(B_r - pr_m), \tag{3.2.30}$$

which can be substituted into the equation

$$G(m) = \frac{1}{2h^2}\left((F(r_{m+1}) + F(r_m))(r_{m+1} - r_m) - (F(r_m) + F(r_{m-1}))(r_m - r_{m-1})\right).$$

It is straightforward to see that the partial derivatives of r_{m+1} are given by

$$\frac{\partial r_{m+1}}{r_m} = -2h\left(\frac{pF(r_m) + (B_r + pr_m)F'(r_m)}{(F(r_m))^2}\right),$$
$$\frac{\partial r_{m+1}}{r_{m-1}} = 1,$$

which are used in the Jacobian entries

$$J(m, m-1) = \frac{1}{2h^2}\Bigg(F'(r_{m+1})\frac{\partial r_{m+1}}{r_{m-1}}(r_{m+1} - r_m) - F'(r_{m-1})(r_m - r_{m-1}) + (F(r_{m+1}) + F(r_m))\frac{\partial r_{m+1}}{\partial r_{m-1}} + (F(r_m) + F(r_{m-1}))\Bigg),$$

and

$$J(m, m) = \frac{1}{2h^2}\Bigg((F'(r_{m+1})\frac{\partial r_{m+1}}{r_m} + F'(r_m))(r_{m+1} - r_m) - F'(r_m)(r_m - r_{m-1}) + (F(r_{m+1}) + F(r_m))(\frac{\partial r_{m+1}}{\partial r_m} - 1) - (F(r_m) + F(r_{m-1}))\Bigg).$$

The remainder of the Newton iteration proceeds as usual.

Now we consider Ω_2. On $\Omega_2 = [\alpha, 1]$, we must discretize the left boundary condition

$$F(r_2^{(n)}(\alpha))\frac{dr_1^{(n)}(\alpha)}{d\xi} - pr_2^{(n)}(\alpha) = B_l. \tag{3.2.31}$$

Let $r_2 = r$, $\xi_2 = \xi$ and $m_2 = m$ for convenience such that $x_1 = \alpha$. Here, (3.2.31) is approximated by

$$F(r_1)\left(\frac{r_2 - r_0}{2h}\right) - pr_1 = B_l, \tag{3.2.32}$$

giving

$$r_0 = r_2 - \frac{2h}{F(r_1)}(B_l + pr_1), \tag{3.2.33}$$

which can be substituted into the equation

$$G(m) = \frac{1}{2h^2}\left((F(r_2) + F(r_1))(r_2 - r_1) - (F(r_1) + F(r_0))(r_1 - r_0)\right).$$

It is straightforward to see that the partial derivatives are given by

$$\frac{\partial r_0}{r_1} = -2h\left(\frac{pF(r_1) + (B_l + pr_1)F'(r_1)}{(F(r_1))^2}\right),$$
$$\frac{\partial r_0}{r_2} = 1,$$

which are used in the Jacobian entries

$$J(1,1) = \frac{1}{2h^2}\Bigg(F'(r_1)(r_2 - r_1) - (F(r_2) + F(r_1)) - \left(F'(r_1) + F'(r_0)\frac{\partial r_0}{\partial r_1}\right)(r_1 - r_0) - (F(r_1) + F(r_0))\left(1 - \frac{\partial r_0}{\partial r_1}\right)\Bigg),$$

and

$$J(1,2) = \frac{1}{2h^2}\Bigg(F'(r_2)(r_2 - r_1) + (F(r_2) + F(r_1)) - F'(r_0)\frac{\partial r_0}{\partial r_2}(r_1 - r_0) + (F(r_1) + F(r_0))\frac{\partial r_0}{\partial r_2}\Bigg).$$

The remainder of the Newton iteration proceeds as usual.

Analysis on two subdomains

To prove convergence of (3.2.24), we provide a slight adaptation of the analysis in [18].

Lemma 3.2.3. *Under the assumptions of Lemma 3.2.1, the BVP given by*

$$\frac{d}{d\xi}\left(F(r)\frac{dr}{d\xi}\right) = 0, \qquad r(0) = 0, \quad F(r)\frac{dr(\beta)}{d\xi} + pr(\beta) = \gamma_\beta, \tag{3.2.34}$$

has a unique solution for all $p > 0$ given implicitly by

$$\int_0^{r(\xi)} F(\tilde{r})d\tilde{r} = (\gamma_\beta - pr(\beta))\xi, \;\; \xi \in (0, \beta), \tag{3.2.35}$$

where p and γ_β are constants and $\beta \in (0,1)$ is fixed.

Proof. The differential equation and boundary condition at $\xi = 0$ is satisfied by

$$\int_0^{r(\xi)} F(\tilde{r})d\tilde{r} = \mathcal{C}\xi,$$

where $\mathcal{C}$ is chosen to satisfy the Robin condition at $\xi = \beta$. Direct calculation gives $\mathcal{C} = \gamma_\beta - pr(b)$, from which (3.2.35) follows.

To establish the existence and uniqueness of $r(\xi)$ satisfying (3.2.35), we notice that if $\xi = \beta$, then the boundary value $r(\beta)$ is the solution θ to

$$G(\theta) = \beta\gamma_\beta, \tag{3.2.36}$$

where $G(\theta) \equiv \int_0 \theta F(\tilde{r})d\tilde{r} + \beta p\theta$.

Under the assumptions of Lemma (3.2.1), G is continuous and uniformly monotonic such that there exists a constant $G_p > 0$ such that $\frac{dG}{d\theta} = F(\theta) + \beta p \geq G_p > 0$. So (3.2.36) has a unique solution $r(\beta)$. The existence of a unique, continuously differentiable solution $r(\xi)$ for $\xi \in (0, \beta)$ follows from considering (3.2.35) with the newly specified $r(\beta)$. Noting that

$$\tilde{G}(\theta) = \int_0 \theta F(\tilde{r})d\tilde{r}, \tag{3.2.37}$$

a continuous and uniformly monotonic function, it follows that it has a continuously differentiable inverse. $\square$

Lemma 3.2.4. *Under the assumptions of Lemma 3.2.1, the BVP given by*

$$\frac{d}{d\xi}\left(F(r)\frac{dr}{d\xi}\right) = 0, \qquad F(r)\frac{dr(\beta)}{d\xi} - pr(\beta) = \gamma_\beta, \quad r(1) = 1, \tag{3.2.38}$$

has a unique solution for all $p > 0$ *given implicitly by*

$$\int_{r(\xi)}^1 F(\tilde{r})d\tilde{r} = (\gamma_\beta + pr(\beta))(1 - \xi), \;\; \xi \in (\beta, 1). \tag{3.2.39}$$

Proof. Equation (3.2.39) follows from direct calculation. To establish existence and uniqueness of a function $r(\xi)$ satisfying (3.2.39), we evaluate this equation for $\xi = \beta$ to obtain

$$\int_{r(\beta)}^1 F(\tilde{r})d\tilde{r} = (1 - \beta)(\gamma_\beta + pr(\beta)),$$

or, after rearranging,

$$\int_{r(\beta}^{1} F(\tilde{r})d\tilde{r} - (1-\beta)(pr(\beta)) = (1-\beta)\gamma_\beta.$$

If we define

$$G(\theta) = \int_{\theta}^{1} F(\tilde{r})d\tilde{r} - (1-\beta)p\theta, \tag{3.2.40}$$

under the assumptions of Lemma 3.2.1, G is continuous and uniformly monotonic and therefore invertible. We conclude that there exists a unique $r(\beta)$ given by $G^{-1}((1-\beta)\gamma_\beta)$. Having determined $r(\beta)$, the unique solution $r(\xi)$ for $\xi \in (\beta, 1)$ is given by

$$\tilde{G}^{-1}((\gamma_\beta + pr(\beta))(1-\xi)), \qquad \text{where} \qquad \tilde{G}(\theta) = \int_{\theta}^{1} F(\tilde{r})d\tilde{r},$$

which is clearly a continuous and uniformly decreasing function under the stated assumptions. □

Theorem 3.2.2. *Consider a monitor function F satisfying the conditions of Lemma 3.2.1. The subdomain solutions on Ω_1 and Ω_2 are given implicitly by*

$$\int_{0}^{r_1^{(n)}(\xi)} F(\tilde{r})d\tilde{r} = R_1(r_1^{(n)}(\alpha))\xi \quad \text{and} \quad \int_{r_2^{(n)}(\xi)}^{1} F(\tilde{r})d\tilde{r} = R_2(r_2^{(n)}(\alpha))(1-\xi), \tag{3.2.41}$$

where the operators R_1 and R_2 are given by

$$R_1(x) = \frac{1}{\alpha}\int_{0}^{x} F(\tilde{r})d\tilde{r} \quad \text{and} \quad R_2(x) = \frac{1}{1-\alpha}\int_{x}^{1} F(\tilde{r})d\tilde{r}. \tag{3.2.42}$$

Proof. The transmission conditions force the operator values to satisfy the recurrence relations

$$R_1(r_1^{(n+1)}(\alpha)) + pr_1^{(n+1)}(\alpha) = R_2(r_2^{(n)}(\alpha)) + pr_2^{(n)}(\alpha), \tag{3.2.43}$$

and

$$R_2(r_2^{(n+1)}(\alpha)) - pr_2^{(n+1)}(\alpha) = R_1(r_1^{(n)}(\alpha)) - pr_1^{(n)}(\alpha). \tag{3.2.44}$$

□

Here, (3.2.43)–(3.2.44) is a nonlinear Peaceman-Rachford type iteration [44]. The convergence of the optimized Schwarz iteration follows from the analysis of similar (nonlinear) Peaceman-Rachford iterations.

The following result was first presented in [18].

Theorem 3.2.3. *Under the assumptions of Lemma* (3.2.1)*, the iteration* (3.2.43)–(3.2.44) *converges globally to the exact solution* $r(\alpha)$ *for all* $p > 0$*. Additionally, the optimized Schwarz iteration has the convergence estimate*

$$||r - r_1^{(2n+1)}||_\infty \leq \rho^n \frac{c_1}{c_0} \cdot \frac{p + \frac{1}{\alpha}c_1}{p + \frac{1}{\alpha}c_0} \rho_{robin}^n |r(\alpha) - r_1^{(0)}(\alpha)|, \tag{3.2.45}$$

$$||r - r_2^{(2n+1)}||_\infty \leq \rho^n \frac{c_1}{c_0} \cdot \frac{p + \frac{1}{\alpha}c_1}{p + \frac{1}{1-\alpha}c_0} \rho_{robin}^n |r(\alpha) - r_2^{(0)}(\alpha)|, \tag{3.2.46}$$

where an upper bound for the contraction factor is

$$\rho_{robin} = \sqrt{\frac{p^2 + \frac{c_1^2}{(1-\alpha)^2} - 2p\frac{c_0}{1-\alpha}}{p^2 + \frac{c_1^2}{(1-\alpha)^2} + 2p\frac{c_0}{1-\alpha}}} \cdot \sqrt{\frac{p^2 + \frac{c_1^2}{\alpha^2} - 2p\frac{c_0}{\alpha}}{p^2 + \frac{c_1^2}{\alpha^2} + 2p\frac{c_0}{\alpha}}} \tag{3.2.47}$$

Proof. A simple rewrite of the recurrence relations (3.2.43) and (3.2.44) gives

$$\begin{aligned}(pI + R_1)r_1^{(n+1)}(\alpha) &= (p_1 + R_2)r_2^{(n)}(\alpha),\\ (pI - R_2)r_2^{(n)}(\alpha) &= (p_1 - R_1)r_1^{(n-1)}(\alpha).\end{aligned}$$

For the operators R_1 and R_2 given by (3.2.42), we have

$$R_1'(r) = \frac{1}{\alpha}F(r) \geq \frac{1}{\alpha}c_0 > 0,$$

and

$$-R_2'(r) = \frac{1}{1-\alpha}F(r) \geq \frac{1}{1-\alpha}c_0 > 0,$$

such that R_1 and $-R_2$ are continuous and uniformly monotonic. Since $p > 0$, the same is true for $pI + R_1$ and $pI - R_2$ and hence they are also invertible; this means that $r_2^{(n)}(\alpha)$ and $r_1^{(n+1)}(\alpha)$ are well defined.

Eliminating $r_2^{(n)}(\alpha)$, we obtain the recursion formula

$$r_1^{(n+1)}(\alpha) = Gr_1^{(n-1)}(\alpha), \tag{3.2.48}$$

where

$$G \equiv (pI + R_1)^{-1}(pI + R_2)(pI - R_2)^{-1}(pI - R_1), \tag{3.2.49}$$

since $pI + R_1$ and $pI + R_1$ are invertible. G can be rewritten as

$$G = (pI + R_1)^{-1} G_1 G_2 (pI + R_1),$$

where we define $G_1 \equiv (pI + R_2)(pI - R_2)^{-1}$ and $G_2 \equiv (pI - R_1)(pI + R_1)^{-1}$. G_1 and G_2 are strict contractions for all $p > 0$, as R_1 and $-R_2$ are uniformly monotonic and Lipschitz continuous. Therefore, the iteration

$$z^{(n)}(\alpha) = G_1 G_2 z^{n-2}(\alpha), \qquad z^{(0)}(\alpha) = (p_1 + R_1) r_1^{(0)}(\alpha),$$

is convergent. Further, since $z^{(2n)}(\alpha) = (pI + R_1) r_1^{(2n)}(\alpha)$, then $r_1^{(2n)}(\alpha)$ is also convergent; we say that $r_1^{(2n)}(\alpha)$ converges to a limit $r_1^*(\alpha)$. It can also be shown that $r_1^{(2n+1)}(\alpha)$ converges to a limit $r_1^*(\alpha)$ as well. Analogously, $r_2^{(2n)}(\alpha)$ and $r_2^{(2n+1)}(\alpha)$ converge to the limit $r_2^*(\alpha)$.

The limit points $r_1^*(\alpha)$ and $r_2^*(\alpha)$ must satisfy (3.2.43) and (3.2.44). We add (3.2.43) and (3.2.44) and find that $r_1^*(\alpha)$=$r_2^*(\alpha) \equiv r^*(\alpha)$. By subtracting (3.2.43) and (3.2.44) we find that $r^*(\alpha)$ satisfies $R_1(r^*(\alpha)) = R_2(r^*(\alpha))$ such that

$$\frac{1}{\alpha} \int_0^{r^*(\alpha)} F(\tilde{r}) d\tilde{r} = \frac{1}{\alpha - 1} \left(\int_0^{r^*(\alpha)} F(\tilde{r}) d\tilde{r} - C \right).$$

We can easily show that $r^*(\alpha) = r(\alpha)$.

We determine ρ_{robin} by computing the Lipschitz constant of the operator $G_1 G_2$ by multiplying the Lipschitz constants of G_1 and G_2. The convergence factor of $r_1^{(n)}(\alpha)$ is related to ρ_{robin} like

$$|r^*(\alpha) - r_1^{(2n)}(\alpha)| \leq L\tilde{L} \rho_{robin}^n |r^*(\alpha) - r_1^{(0)}(\alpha)|,$$

where L and $\tilde{L}$ are the Lipschitz constants for $(pI + R_1)^{-1}$ and $(pI + R_1)$, respectively. We can show that $L = (p + \frac{1}{\alpha} c_0)^{-1}$ and $\tilde{L} = (p + \frac{1}{\alpha} c_0)$. Using this and the fact that $|r_1^{(2n)}(\xi) - r(\xi)| \leq \frac{c_1}{c_0} |r(\alpha) - r_1^{2n}(\alpha)$, we have

$$|r_1^{(2n)}(\xi) - r(\xi)| \leq \frac{c_1}{c_0} \cdot \frac{p + \frac{1}{\alpha} c_1}{p + \frac{1}{\alpha} c_0} \rho_{robin}^n |r(\alpha) - r_1^{(0)}(\alpha)|. \tag{3.2.50}$$

The estimate on the second subdomain follows accordingly. □

Analysis of contraction factor

Given (3.2.47), we can determine an analytic, problem dependent constant p that minimizes the upper bound of the contraction factor ρ_{robin}.

Theorem 3.2.4. *By simple calculus, we determine that*

$$p_{upper} = \sqrt{\frac{c_1^2}{(1-\alpha)\alpha}}, \tag{3.2.51}$$

minimizes the upper bound of the contraction factor ρ_{robin}.

Remark 3.2.1. *Since* $0 < \alpha < 1$ *and* $0 < c_0 \le F(r) \le c_1 < \infty$, (3.2.51) *will be positive as desired. We remark that the constant* c_0 *has no effect on* (3.2.51).

In practice, the constant obtained analytically from (3.2.51) generally will not be equivalent to the optimal constant to minimize the iteration count numerically. However, as we decrease the mesh width $\Delta x \to 0$, we expect the discretized results to converge to the continuous result. That is, we expect the optimal constant to converge to (3.2.51) as the mesh width decreases.

Consider the example

$$x(r) = \tanh(r) + 2r, \quad y(r) = r^{10} + r, \quad 0 \le r \le 1. \tag{3.2.52}$$

From (3.2.51) and setting $\alpha = 0.5$ and numerically determining c_0 and c_1, we see that the p value that minimizes the upper bound is $p \approx 23$. However, the actual experimental numerical p_{exp} value for $m = 81$ is shown to be $p \approx 6$. This is shown in Figure 3.8. This large discrepancy between p_{exp} and p_{upper} is not unexpected, as p_{upper} depends on the upper bound ρ_{robin} given by (3.2.47). If this is a loose upper bound, we can not expect accuracy of the analytical optimal p value p_{upper}.

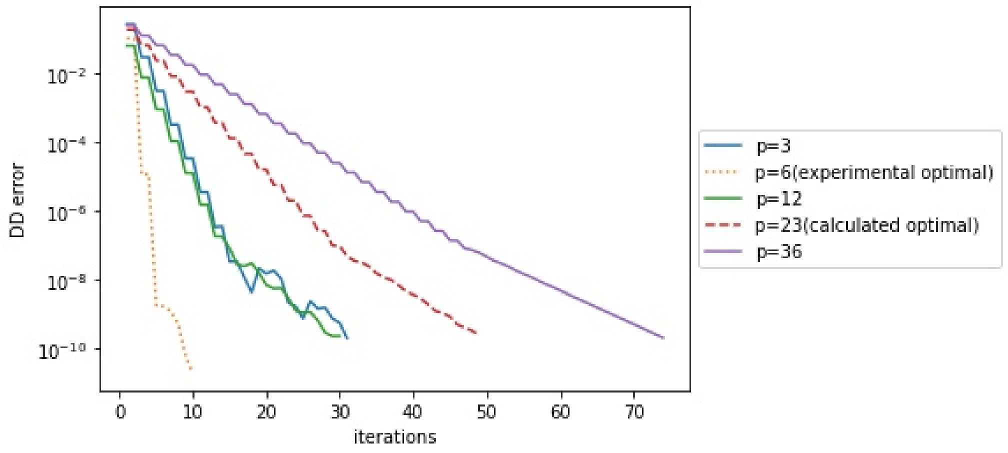


Figure 3.8: Plot of iterations versus p for various constants p in the optimized Schwarz iteration (3.2.24). The DD error is shown for the parallel DD iteration (3.2.2) applied to the curve (3.2.52), where the arc-length-based monitor function (2.2.6) is used. Here, $m = 81$ and the initial guess is given by $r_0 = \xi$. DD error refers to the error between the DD solution and the single domain solution.

Another way to determine p_{exp} for each curve is to implement a "brute force" approach, evaluating the iterations for multiple p values to determine the nature of the iterations versus p figures. This is shown for $m = 81$ and $m = 161$ nodes in Figure 3.9. We can see from Figure 3.9 that while the number of iterations slightly changes with m, we generally obtain similar optimal p results for various m values. This tells us that we can use a small number of nodes to determine the optimal p value.

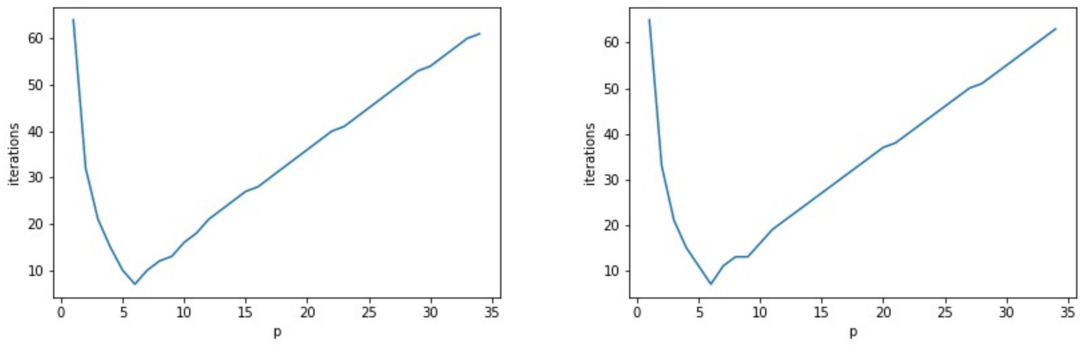


Figure 3.9: Plot of iterations versus p for $m = 81$ nodes (left) and $m = 161$ nodes (right) in the optimized Schwarz iteration (3.2.24) for the curve (3.2.52), where the arc-length-based monitor function (2.2.6) is used. The initial guess is given by $r_0 = \xi$.

3.2.3 Comparison of DD methods

We provide numerical results directly comparing the optimized Schwarz methods for various p constants with classical Schwarz in Figure 3.10. We see from Figure 3.10 that the Robin transmission conditions obtained from using optimized Schwarz results in convergence in a lower number of iterations than classical Schwarz. We see that increasing p in the optimized iteration results in less efficient convergence. In Figure 3.10, $p = 8$ produces the more efficient convergence in terms of iterations.

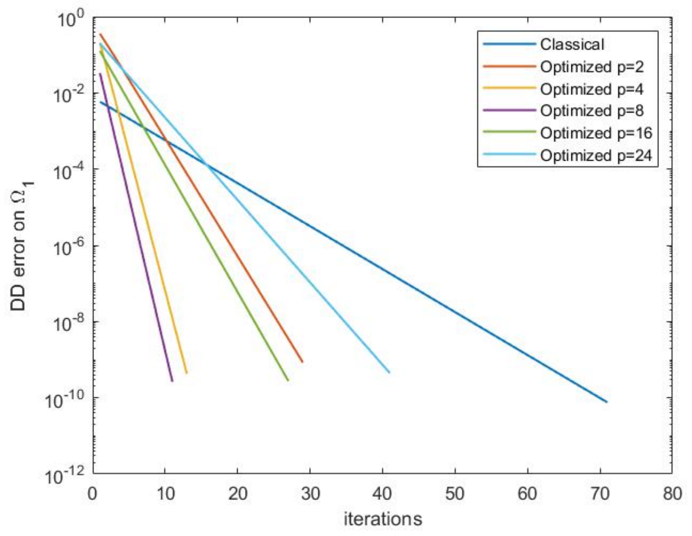


Figure 3.10: A plot of DD error versus iterations (semi-log scale) comparing the parallel classical Schwarz iteration (3.2.2) to the optimized Schwarz iteration (3.2.24) for various constants p on $S = 2$ subdomains. Results are given for the ellipse (2.2.14), where the curvature-based monitor function (2.2.7) is used. Here, $m = 64$ nodes are equidistributed and the initial guess is given by $r_0 = \xi$. DD error refers to the error between the DD solution and the single domain solution.

Chapter 4

The Single Domain Equidistribution for the Solutions of Differential Equations on Static Curves

In this chapter, we explore equidistribution of $u(r)$ on a curve $\mathbf{x}(r)$, where $u(r)$ is the unknown solution of a differential equation. In this case, we employ an alternating method, alternating between mesh solves and physical PDE solves. Numerical results are provided throughout.

4.1 The Single Domain Solution of Static Boundary Layer Problems

Up until this point, we have focused on single domain and multi-domain approaches to the equidistribution of a known function $u(r)$ on a curve. We now continue by using r-refinement to equidistribute a function $u(r)$ on $\mathbf{x}(r)$, where u is an unknown solution to a differential equation. For our example problems, we look to numerically approximate the solution to boundary layer problems of the form

$$-\epsilon\Delta_{\mathcal{C}}u+\mathbf{k}\cdot\nabla_{\mathcal{C}}u+cu=f, \quad u \text{ on } \mathcal{C}, \quad 0<\epsilon<<1, \qquad u(0)=\alpha,\ u(b)=\beta, \tag{4.1.1}$$

where $\Delta_{\mathcal{C}}$ and $\nabla_{\mathcal{C}}$ denote the surface Laplace and gradient operators, respectively, and $\mathcal{C}$ denotes the surface on which the problem is defined. The surface Laplacian operator is commonly referred to as the Laplace-Beltrami operator and is a generalization of the typical Laplace operator for curved geometries. These operators are necessary as u is posed on the curve.

These boundary layer problems are useful to us as they may result in large/rapid changes in the function u. When equation (4.1.1) is discretized using certain types of discretizations with a mesh that does not have a sufficiently large number of nodes, the resulting discrete numerical solution can exhibit rapid fluctuations. For more information on boundary layer problems of the form (4.1.1), see [24].

To solve the boundary layer problem in the case where u is posed on the curve $\mathbf{x}(r)$, we must first simplify the differential operators. The generalized Laplace-Beltrami operator is given by

$$\Delta_{\mathcal{C}} u = |g|^{-1/2} \partial_i (\sqrt{|g|} g^{ij} \partial_j u),$$

where g denotes the metric tensor, $|g|$ denotes the determinant of g, and g^{ij} denotes the i, jth entry of the inverse of the metric tensor, g^{-1}. By definition, the components of the metric tensor are given by

$$g_{ij} = g\left(\frac{\partial}{\partial r^i}, \frac{\partial}{\partial r^j}\right),$$

for the n dimensional coordinate system $(r^1, \ldots, r^n)$.

We want to explicitly discretize (4.1.1) on a static parametric curve given by $\mathbf{x}(r) \in \mathbb{R}^2$ or $\mathbf{x}(r) \in \mathbb{R}^3$. Here, $\mathcal{C}$ is a one-dimensional surface, as the curve is expressed in a one-dimensional coordinate system r^1 such that $r^1 = r$ is a one-dimensional parameter for the curve. On a one-dimensional surface $\mathcal{C}$, this gives the metric tensor as a single element g_{11} such that

$$g_{11} = g\left(\frac{\partial}{\partial r^1}, \frac{\partial}{\partial r^1}\right) = g\left(\frac{\partial}{\partial r}, \frac{\partial}{\partial r}\right).$$

We use the induced metric, also known as first fundamental form of the curve. The first fundamental form is defined as the inner product of the tangent vectors of the curve. On a parametric curve $\mathbf{x}(r)$, this becomes the dot product

$$g_{11} = \mathbf{x}'(r) \cdot \mathbf{x}'(r) = |\mathbf{x}'(r)|^2.$$

Hence,

$$|g_{11}| = |\mathbf{x}'(r)|^2,$$

and

$$|g^{11}| = |\mathbf{x}'(r)|^{-2}.$$

This allows us to rewrite the Laplace-Beltrami operator as

$$\begin{aligned}\Delta_{\mathcal{C}} u &= \nabla \cdot (\nabla_{\mathcal{C}} u) \\ &= \frac{1}{|\mathbf{x}'|}\partial_r \left(\frac{1}{|\mathbf{x}'|}\partial_r u \right).\end{aligned}$$

Likewise, the gradient operator can be written as

$$\nabla_{\mathcal{C}} u = \frac{1}{|\mathbf{x}'|}\partial_r u.$$

Using these expressions, (4.1.1) is equivalent to

$$-\epsilon |\mathbf{x}'(r)|^{-1} \left(|\mathbf{x}'(r)|^{-1} u_r \right)_r + k(r)|\mathbf{x}'(r)|^{-1} u_r + c(r)u = f(r). \tag{4.1.2}$$

We note that on a 2D curve where $\mathbf{x}(r) = (x(r), y(r))$, this becomes

$$-\epsilon \frac{1}{\sqrt{x'(r)^2 + y'(r)^2}} \left(\frac{1}{\sqrt{x'(r)^2 + y'(r)^2}} u_r \right)_r + k(r) \frac{1}{\sqrt{x'(r)^2 + y'(r)^2}} u_r + c(r)u = f(r). \tag{4.1.3}$$

For more details regarding this choice of metric and the differential operators, we suggest [27, 35].

To discretize (4.1.2), one method is to expand the first term using the product rule. Defining

$$|\mathbf{x}'(r)|^{-1} := \psi(r), \tag{4.1.4}$$

this gives

$$-\epsilon\psi \left(\psi_r u_r + \psi u_{rr} \right) + k\psi u_r + cu = f(r), \qquad u(0) = \alpha, \;\; u(b) = \beta. \tag{4.1.5}$$

We note that we could also discretize (4.1.2) directly without expanding by the product rule.

To combine equidistribution with the solution of a differential equation, we need to solve (4.1.5) on a non-uniform mesh. We would generally discretize the first and second derivatives using the classical centered finite difference operators for non-uniform meshes, given by

$$\partial_r(u(r_i)) \approx \frac{h_{i-1}^2(U_{i+1}-U_i)+h_i^2(U_i-U_{i-1})}{h_{i-1}h_i(h_{i-1}+h_i)} \tag{4.1.6}$$

and

$$\partial_r^2(u(r_i)) \approx \frac{2[h_{i-1}(U_{i+1}-U_i)-h_i(U_i-U_{i-1})]}{h_{i-1}h_i(h_{i-1}+h_i)} \tag{4.1.7}$$

for index $i = 2, \dots, m-1$, where $U = (U_0, U_1, \dots, U_m)^T$ is the discrete approximation of u on the curve and $h_i = r_{i+1} - r_i$. We note that on a uniform mesh (when $h_i = h_{i-1}$), (4.1.6) and (4.1.7) simplify to the standard centered difference approximations. However, on a non-uniform mesh, a Taylor series analysis shows that the second derivative approximation (4.1.7) is only a first order approximation to $u''(r_i)$ when $h_i \neq h_{i-1}(1+O(h_i))$. However, as is common with finite difference approximations on a non-uniform mesh, (4.1.7) will often produce a second order approximation when $\max_i h_i$ is sufficiently small in the neighborhood of the layer [24]. There are methods that can be used to improve the accuracy of the second derivative approximation such as introducing a staggered mesh, but for simplicity we will use (4.1.7) throughout.

Before we state any iterations to solve (4.1.5), we clarify some terminology to be used in following sections.

Given a general uniform computational domain Ω given by $\xi \in [\alpha, \beta]$ and Dirichlet boundary values r_α and r_β, and given a monitor function $F(r)$, we approximate the solution of

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) = 0, \qquad r(\alpha) = r_\alpha,\;\; r(\beta) = r_\beta, \tag{4.1.8}$$

iteratively here via Newton's method

$$r^{(n+1)} = r^{(n)} + \delta,$$

where $r = (r_0, \dots, r_m)^T$ is a mesh of size $m+1$ and r_i approximates $r(\xi_i) = r(i/m)$. For Newton's method, (4.1.8) is discretized with the scheme (2.2.12) provided in Section

2.2, giving

$$G_i = \frac{\left(F(r_{i+1}) + F(r_i)\right)\left(r_{i+1} - r_i\right) - \left(F(r_i) + F(r_{i-1})\right)\left(r_i - r_{i-1}\right)}{2\Delta\xi^2},$$

for $i = 1, 2, \ldots, m-1$, and the corresponding Jacobian

$$J_{ij}(U) = \frac{\partial}{\partial U_j} G_i(U),$$

for $i, j = 1, 2, \ldots, m-1$. Solving the nonlinear equations obtained from discretizing (4.1.8) with Newton's method will be called a "mesh solve", which we will label as an "M" solve on Ω moving forward.

Consider a general domain $[r_\alpha, r_\beta]$ discretized with a mesh of size $m+1$ given by $r = (r_0, \ldots, r_m)^T$, with $r_{i-1} < r_i < r_{i+1}$, $r_0 = r_\alpha$, and $r_m = r_\beta$. This notation is used to define a general domain r with endpoints r_α and r_β; in practice, this could be the single domain or a subdomain. Additionally, consider general Dirichlet boundary values given by $\mathcal{B}_1$ and $\mathcal{B}_2$. We approximate the solution of

$$-\epsilon\psi\left(\partial_r\psi\partial_r u + \psi\partial_r^2 u\right) + k\psi\partial_r u + cu = f, \qquad u(r_\alpha) = \mathcal{B}_1,\ \ u(r_\beta) = \mathcal{B}_2, \tag{4.1.9}$$

by forming and solving

$$AU = f, \tag{4.1.10}$$

where $f = (\mathcal{B}_1, f(r_1), \ldots, f(r_{m-1}), \mathcal{B}_2)^T$ and

$$A = \begin{bmatrix} \eta_0 & \gamma_0 & & & & & & \\ \phi_1 & \eta_1 & \gamma_1 & & & & & \\ & \phi_2 & \eta_2 & \gamma_2 & & & & \\ & & \phi_3 & \eta_2 & \gamma_3 & & & \\ & & & \phi_4 & \eta_4 & \gamma_4 & & \\ & & & & \phi_5 & \ddots & \ddots & \\ & & & & & \ddots & \ddots & \gamma_{m-1} \\ & & & & & & \phi_m & \eta_m \end{bmatrix}.$$

Here, the discretization of (4.1.9) using finite difference methods (4.1.6) and (4.1.7) results in the matrix entries

$$
\begin{aligned}
\phi_{i+1} &= \frac{\epsilon\psi_{i+1}(h_i^2(\psi_{i+2}-\psi_{i+1})+h_{i+1}^2(\psi_{i+1}-\psi_i))h_{i+1}^2}{(h_{i+1}^2h_i^2(h_{i+1}+h_i)^2)} - \frac{2\epsilon\psi_{i+1}^2h_{i+1}}{h_ih_{i+1}(h_i+h_{i+1})} \\
&\quad - \frac{k_{i+1}\psi_{i+1}h_{i+1}^2}{h_ih_{i+1}(h_i+h_{i+1})}, \\
\eta_{i+1} &= \frac{\epsilon\psi_{i+1}(h_i^2(\psi_{i+2}-\psi_{i+1})+h_{i+1}^2(\psi_{i+1}-\psi_i)(h_i^2-h_{i+1}^2)}{(h_{i+1}^2h_i^2(h_{i+1}+h_i)^2)} + \frac{2\epsilon\psi_{i+1}^2(h_i+h_{i+1})}{h_ih_{i+1}(h_i+h_{i+1})} \\
&\quad - \frac{k_{i+1}\psi_{i+1}(h_i^2-h_{i+1}^2)}{h_ih_{i+1}(h_i+h_{i+1})} + c_{i+1}, \\
\gamma_{i+1} &= -\frac{\epsilon\psi_{i+1}(h_i^2(\psi_{i+2}-\psi_{i+1})+h_{i+1}^2(\psi_{i+1}-\psi_i))h_i^2}{(h_{i+1}^2h_i^2(h_{i+1}+h_i)^2)} - \frac{2\epsilon\psi_{i+1}^2h_i}{h_ih_{i+1}(h_i+h_{i+1})} \\
&\quad + \frac{k_{i+1}\psi_{i+1}h_i^2}{h_ih_{i+1}(h_i+h_{i+1})},
\end{aligned}
$$

for $i = 0, \ldots, m-2$. To complete the system, we set $\eta_0 = 1, \gamma_0 = 0, \phi_m = 0$, and $\eta_m = 1$. Here, $\psi_i = \psi(r_i), k_i = k(r_i), c_i = c(r_i)$, and $h_i = r_i - r_{i-1}$. Approximating (4.1.9) with the above discretization and then solving (4.1.10) will be called a "physical solve", which we will label as a "P" solve on the given domain. We note that this discretization arises from the centered difference approximation, but one could similarly employ an upwind approximation.

Given an initial (often uniform) initial mesh $r^{(0)}$ on which the curve is discretized, the proposed iterative procedure to solve boundary layer problems on a curve is given in Algorithm 1.

Algorithm 1: Single domain MP method on a curve

```
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
initialize the uniform mesh $r^{(0)}$;
Complete a physical solve on $r^{(0)}$ to obtain $U^{(0)}$;
for $n = 0, 1, \ldots,$ do
    Given the monitor function vector $F$ (which uses $U^{(n)}$), complete a mesh
      solve on the computational coordinate domain $\Omega = [0, 1]$ with boundary
      values $r_0 = r(0)$ and $r_m = r(1)$ to determine the new grid $r^{(n+1)}$. $F$ must
      be interpolated onto the new grid at each step of the equidistribution;
    Complete a physical solve with boundary values $\mathcal{B}_1$ and $\mathcal{B}_2$ on $r^{(n+1)}$ to
      obtain $U^{(n+1)}$;
    if $\max(||U^{(n)} - U^{(n-1)}||_\infty, ||r^{(n)} - r^{(n-1)}||_\infty) < tol$ then
        BREAK LOOP;
    end
end
```

Remark 4.1.1. *As we are using a Newton iteration to solve the mesh BVP, we must numerically compute the derivative of the monitor function F to use in the Jacobian J. We interpolate the discrete approximations of the monitor function F using a cubic spline interpolation. The differentiability of a cubic spline allows us to directly obtain F' from the spline coefficients, rather than performing the finite difference approximation* (4.1.6) *on the interpolated vector F. This allows for a more accurate computation of $\partial_r F$.*

Additionally, in the case where we begin a mesh solve by interpolating the previous physical solve U from the previous mesh, we would use the same process to obtain U'. This U' may need to be calculated once in order to determine F, depending on the monitor function. More details on the choice of monitor function is given in Section 2.4.

We refer to these methods as “MP” methods, as in [51] but with slightly different notation. This denotes the alternating of the mesh and physical solves.

We proceed with an example, computing the solution of (4.1.5) with $k(r) = -1, c(r) = 0$, and $f(r) = -1$; that is, we solve

$$\epsilon\psi\left(\psi_r u_r + \psi u_{rr}\right) + \psi u_r = 1, \quad u \text{ on } \mathcal{C}, \quad u(0) = u(1) = 1, \tag{4.1.11}$$

where $\mathcal{C}$ is given by the ellipse

$$x(r) = A\cos(2\pi r), \quad y(r) = B\sin(2\pi r), \quad 0 \leq r \leq 1. \tag{4.1.12}$$

We note that in this example, the function $u(r)$ requires equidistribution much more than the curve $\mathbf{x}(r)$. Plots of the numerical solution of (4.1.11) with and without equidistribution are shown in Figure 4.1. We note since the boundary layer problem (4.1.11) has its layer (and therefore needs equidistribution the most) around $r \approx 0$, equidistribution concentrates the nodes in this area. Concentrating the nodes around $r \approx 0$ on the ellipse with radius $A = 3$ and $B = 0.5$ corresponds to a higher concentration of nodes in the regions of $x \approx 3$ and $y \approx 0$. This is seen in Figure 4.1. We also note from Figure 4.1 that the uniform mesh does not resolve the layer, the centered difference approximation gives jagged "wiggles" in the layer. Equidistribution is a way to resolve this issue.

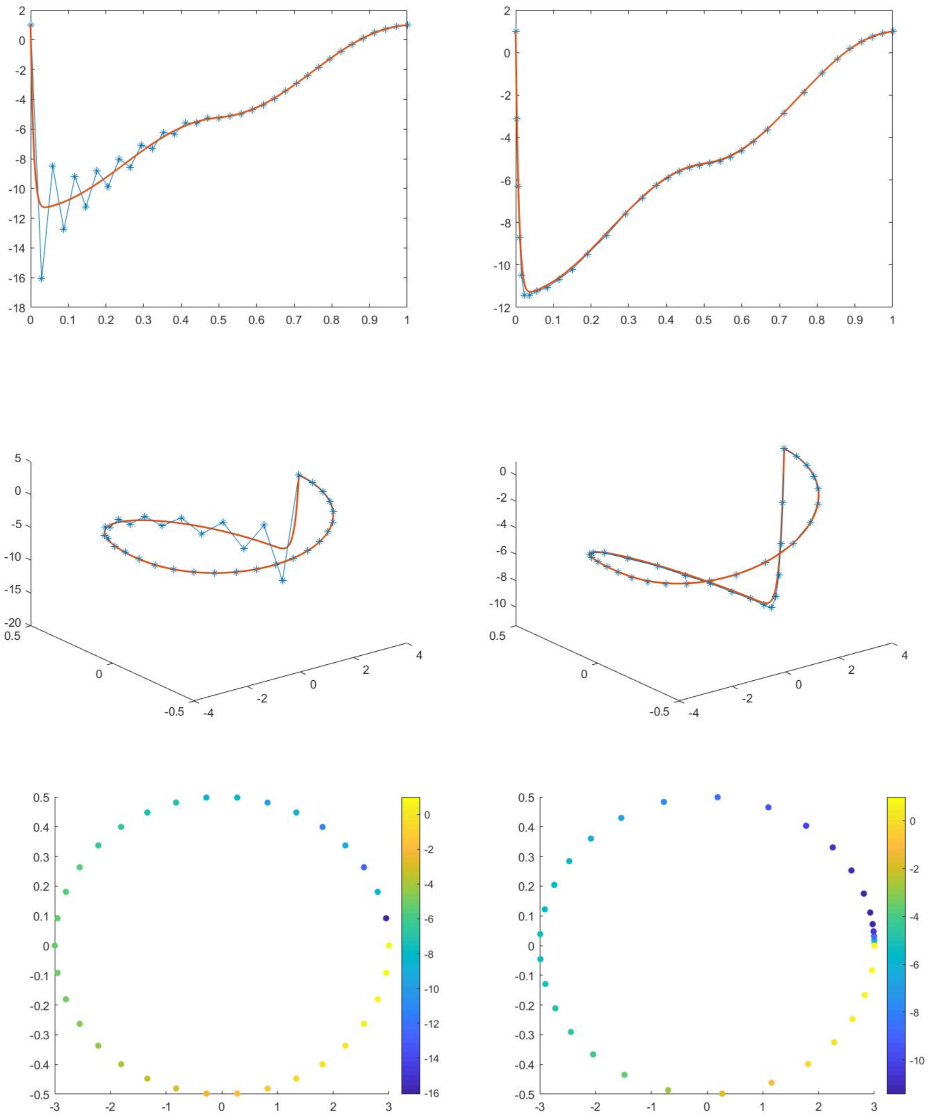


Figure 4.1: Solutions of (4.1.11) with $\epsilon = 0.025$ posed on the ellipse (4.1.12) with $A = 3, B = 0.5$ and $m = 30$ nodes after $n = 3$ iterations of Algorithm 1. Solutions are given both on a uniform mesh with no equidistribution (left) and on a mesh equidistributed with monitor function (2.4.6) (right). Top: 2d plot of discrete solution U versus r, plotted with the fine grid solution shown in solid red for comparison. Row 2: 3d plot of discrete solution U on $\mathbf{x}(r)$, plotted with the fine grid solution shown in solid red for comparison. Bottom: Plot of discrete solution U on $\mathbf{x}(r)$, where the color of a point on the curve represents the numerical value of U.

Remark 4.1.2. *Since we are working with a small number of uniformly spaced nodes and a centered discretization, the numerical solution U will be generally be nonsmooth.*

This will lead to a nonsmooth monitor function F, which can cause Newton's method to fail to converge. If necessary, this is often combated by smoothing the monitor function at each iteration using the method discussed in [30]; this approach computes

$$\tilde{F}_i = \sqrt{\sum_{\min(1,i-p)}^{\max(N,i+p)} (F_r)^2 \left(\frac{\gamma}{1+\gamma}^{|k-i|}\right) \Big/ \sum_{\min(1,i-p)}^{\max(N,i+p)} \left(\frac{\gamma}{1+\gamma}^{|k-i|}\right)}, \tag{4.1.13}$$

$i = 1, \ldots, N$. *Here,* $0 < \gamma < 1$ *and* $p > 1$ *are chosen smoothing parameters. A higher p value corresponds to a smoother monitor function. We then complete a mesh solve via Newton's method using* $\tilde{F}$ *in place of* F *at each iteration. We note that the authors of [30] remarked that choosing the smoothing parameter as* $p = 1$ *or* $p = 2$ *is often sufficient.*

An example of the effect smoothing has on the monitor function is shown in Figure 4.2. We see that the result without smoothing is a more jagged, less smooth monitor function F. We note that in the boundary layer case, smoothing will only be necessary in the first few alternating iterations, as that is when the physical discrete solution U will be jagged in nature. As the alternating procedure converges, smoothing becomes less necessary. This is due to the solution of the boundary layer problem having smoothness in nature. For information on using adaptivity to solve hyperbolic problems with a nonsmooth solution, see [50].

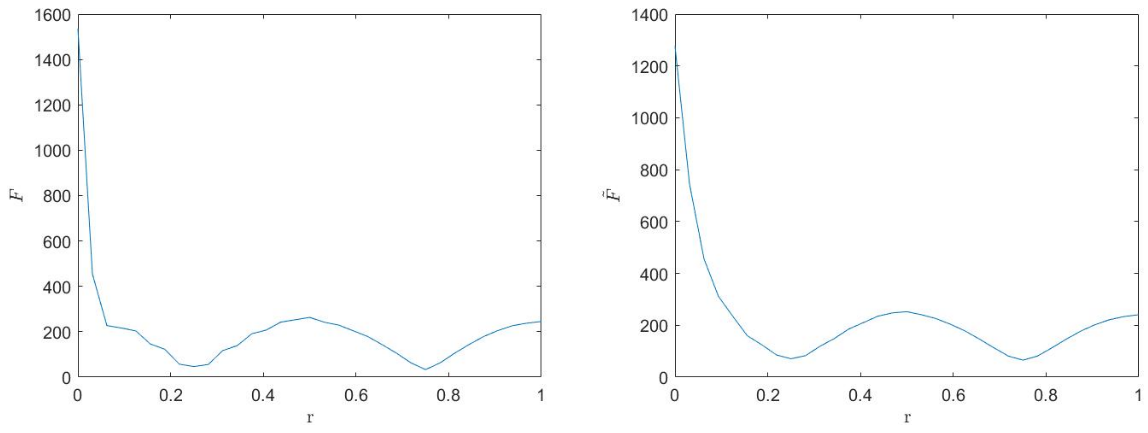


Figure 4.2: A visual of nonsmoothed and smoothed monitor functions with the smoothing equation (4.1.13) and smoothing parameters $p = 4$ and $\gamma = 0.5$. The monitor functions are shown for the mesh (M) solve at alternating iteration $n = 1$ of Algorithm 1 to solve (4.1.11) posed on the ellipse (4.1.12) with $A = 3$ and $B = 0.5$ The non-smooth vector F (left) and the smoothed vector $\tilde{F}$ (right) are formed from the monitor function (2.4.6).

If $\mathcal{C}$ is an interval on the axis, then the Laplace-Beltrami and surface gradient operators simplify to the classic Laplace and gradient operators. This gives a simplified form of (4.1.1) as

$$-\epsilon u'' + k(r)u' + c(r)u = f, \quad u(0) = \alpha, \;\; u(1) = \beta. \tag{4.1.14}$$

In the interval case, we are completing a "modified" physical solve at each iteration, as there are clearly slight adjustments to be made in the discretization. There would be no computation of $\psi(r)$, and we use an accordingly modified matrix tridiagonal matrix A. We also note that the monitor function used during the mesh solve would be different, as there are no curve features to consider.

The procedure for the iteration on a line given an initial mesh $x^{(0)}$, is given in Algorithm 2.

Algorithm 2: single domain MP method on a line

```
Result:
initialize the uniform mesh $x^{(0)}$;
Complete a modified physical solve for $u^{(0)}$ on $x^{(0)}$ ;
for $n = 0, 1, \ldots,$ do
    Given the monitor function vector $F$ (which uses $U^{(n)}$), complete a mesh
     solve on the computational coordinate domain $\Omega = [0, 1]$ to determine
     the new grid $x^{(n+1)}$. At each step of the equidistribution (each Newton
     iteration), interpolate $F$ onto the new grid to eventually give $x^{(n+1)}$ ;
    Complete a modified physical solve on $x^{(n+1)}$ to obtain $U^{(n+1)}$;
    if $\max(||U^{(n)} - U^{(n-1)}||_\infty, ||x^{(n)} - x^{(n-1)}||_\infty) < tol$ then
        BREAK LOOP;
    end
end
```

As a simple example of the line case, we consider the boundary layer problem

$$-\epsilon u'' - u' = -1, \quad u(0) = u(1) = 1, \tag{4.1.15}$$

an example of (4.1.1) with $b(x) = -1, c(x) = 0$, and $f(x) = -1$, where $u(x)$ is posed on the interval $x \in [0, 1]$. On the 1D interval, the surface gradient and Laplacian simplify to the first and second derivatives, respectively.

Plots of the numerical solution of (4.1.15) with and without equidistribution are shown in Figure 4.3. Since we are solving the equation on a line, we use the arc-length monitor function (2.4.2).

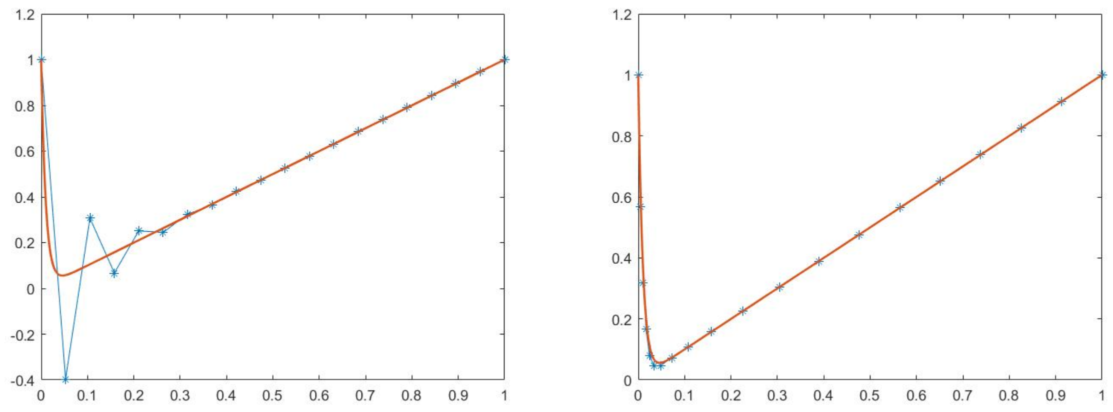


Figure 4.3: Solutions of (4.1.15) with $\epsilon = 0.01$ and with $N = 20$ nodes after $n = 3$ iterations of Algorithm 2. Solutions are given both on a uniform mesh with no equidistribution (left) and on a mesh equidistributed using monitor function (2.4.2) (right). Solution are plotted with the fine grid solution in solid red for comparison.

We now review some theoretical evidence for the convergence of Algorithm 1. In [36], the author considers general linear (and quasi-linear) boundary layer problems and shows the well-posedness and error bounds for the algorithm, working mainly in the discrete case. The relevant results to this paper will be summarized here. It is important to note that there are many small differences between the method used in [36] and the method used here. These differences will be discussed in this section along with corresponding statements on the likelihood of the analysis extending to our algorithm.

The author provides the following two theorems. Theorem 4.1.1 states the accuracy of the solution obtained when the algorithm is terminated from its stopping criteria. When the algorithm is terminated, the computed physical solution is a first order accurate approximation to the true solution. We note that the algorithm used in [36] varies slightly from our algorithm; this is discussed in further detail after the theorems are stated.

Theorem 4.1.1. *Suppose that the algorithm reaches its stopping criterion and halts. Let the resulting mesh be given as x_i^*. Let u_i^* be the discrete solution computed on this mesh, and let $u^*(x)$ be the piecewise linear interpolant of (x_i^*, u_i^*). Then for some constant C independent of the mesh and ϵ, we have*

$$\max_{0 \le x \le 1} |u^*(x) - u(x)| \le CN^{-1}.$$

Theorem 4.1.2 gives an upper bound on the number of iterations k of the algorithm needed for first order accuracy, regardless of the constant ϵ in the boundary layer problem. The solution of the algorithm after k iterations is determined to be a first order accurate approximation to the true solution.

Theorem 4.1.2. *Let N be sufficiently large independent of ϵ and of the number of iterations taken by the algorithm. Then there exists a positive integer k such that $k \leq C' \ln(\frac{1}{\epsilon})/ln(N)$ and $||e^{(K)}||_\infty \leq C'' N^{-1}$, where $e^{(k)}(x) = u^{(k)}(x) - u(x)$ is the error in the solution at iteration k computed by the algorithm. Constants C' and C'' are independent of ϵ and N.*

We now discuss the differences between the method in [36] and the method proposed in this thesis. In [36], the author considers an alternating MP boundary layer problem on the line in 1D. There, instead of a damped Newton method, Kopteva uses a variant of de Boor's algorithm to solve for the mesh, a common algorithm used in 1D moving mesh methods, originally described in [51]. The main advantage of de Boor's algorithm is that it conserves the mesh ordering; we have replaced this with a modified damped Newton method that requires the mesh to remain monotonic throughout each iteration. Therefore, we expect this change in algorithm to have little to no effect on convergence. Specifically, Kopteva states "We expect any other algorithm based on the same or similar monitor function to enjoy similar properties and require a similar amount of iterations."

Kopteva also focuses purely on a monitor function based on arc-length; while this is a reasonable step in the line case, our results have shown that on a curve, equidistributing by just the curve's arc-length can produce a high interpolation error; see Section 2.2 for details. We want to equidistribute in a way that concentrates nodes in areas of both high arc-length of the physical function u and high curvature of the curve $\mathbf{x}(r)$. However, Kopteva suggests that using a curvature-based monitor function on the line can increase the algorithm's accuracy from first order to second order. Therefore, we expect that one would be able to use a monitor function based on the arc-length of the physical function u and the curvature of the curve $\mathbf{x}(r)$ and still obtain convergence of the algorithm. Additionally, Kopteva states that for any reasonable monitor function, both lower and upper bounds on the monitor function should exist, and this results in the existence of the discrete solution of the algorithm.

It is stated in [36] that there are other conditions that should be sufficient to extend their general analysis to other problems and monitor functions. These include a sharp lower bound on the monitor function and constant C_1 such that

$$1 \leq M^{(N)}(x); \qquad \int_0^1 M^{(N)}(x)dx \leq C_1, \tag{4.1.16}$$

where $M^{(N)}(x)$ denotes the linear interpolant of the discrete approximation $M_i, i = 1, \ldots, N$, where N is the total number of mesh nodes.

Lastly, Kopteva considers an upwind method to discretize the physical PDE, compared to the centered difference method considered here. Hence we could modify our method to a upwind differencing scheme or attempt to generalize Kopteva's results to the centered difference case.

For the time being, we refrain from completing an entire analysis of our single domain method, as that is not our focus. We leave the discrete analysis to future work, with the generalizations and framework stated above.

4.2 The Single Domain Solution of Time-Dependent Differential Equations on Static Curves

A natural application of equidistribution on curves occurs when we wish to solve time-dependent PDEs posed on curves. As discussed in the introduction, these problems occur often in real-world applications. In this section, we propose an alternating algorithm to solve time-dependent PDEs on static curves with equidistribution.

Consider the general equation

$$u_t = f(x, u, \nabla_{\mathcal{C}} u, \Delta_{\mathcal{C}} u), \quad u \text{ on } \mathcal{C}, \tag{4.2.1}$$

where $\mathcal{C}$ is the given closed curve, $\nabla_{\mathcal{C}}$ is the surface gradient and $\Delta_{\mathcal{C}}$ is the generalized Laplace-Beltrami operator on a surface, given by

$$\Delta_{\mathcal{C}} u = |g|^{-1/2} \partial_i (\sqrt{|g|} g^{ij} \partial_j u). \tag{4.2.2}$$

As discussed in Section 4.1, on a parametric curve (4.2.1) can be written as

$$u_t = f\left(x, u, |\mathbf{x}'(r)|^{-1} u_r, |\mathbf{x}'(r)|^{-1} \left(|\mathbf{x}'(r)|^{-1} u_r\right)_r\right). \tag{4.2.3}$$

Specifically in this section, we focus on the advection-diffusion equation given by

$$u_t = \nabla_{\mathcal{C}} u + \Delta_{\mathcal{C}} u + f, \tag{4.2.4}$$

where periodic boundary conditions are chosen to be consistent with (4.2.4) being posed on a closed curve. From the discussions in Section 4.1, we can rewrite (4.2.4), posed on the curve $(x(r), y(r))$, as

$$u_t = (\psi(r)\psi_r(r) + \psi(r))u_r + \psi(r)\psi(r)u_{rr} + f(r,t), \quad u(0,t) = u(b,t), \tag{4.2.5}$$

where ψ is given in (4.1.4). When $\mathbf{x}(r) = (x(r), y(r))$, this gives $\psi(r) = \frac{1}{\sqrt{x'(r)^2+y'(r)^2}}$ and $r \in [0, b]$.

As a specific example, we choose the PDE (4.2.5), with $f(r,t)$ chosen so that

$$u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}} \tag{4.2.6}$$

is the true solution. This solution has nearly periodic boundary conditions on the interval $r \in [0, 1]$ if $t \leq 1$. Here, we choose $\epsilon = 0.01$, and choose $f(r,t) = u_t - (\psi(r)\psi_r(r) + \psi(r))u_r - \psi(r)\psi(r)u_{rr}$, where $\psi(r)$ is a function that is determined by the curve $\mathbf{x}(r)$. The initial condition is given by $u(r,0) = e^{-\frac{(r-\frac{1}{4})^2}{\epsilon}}$.

We aim to solve (4.2.5) coupled with a mesh solve. As (4.2.6) gives a single pulse/wave that moves with time, we suspect that using an equidistributed mesh will provide a more accurate solution. This is shown later on with interpolation errors. Combining this with periodic boundary conditions, we have formed the problem to be solved.

Consider the domain $\Omega = r \in [0, b]$ and its corresponding mesh of size $m+1$ given by $r = (r_0, \dots, r_m)^T$, with $r_{i-1} < r_i < r_{i+1}$, $r_0 = 0$, and $r_m = b$. We approximate the solution to (4.2.5) using an implicit backward Euler scheme at each time step. To discretize the periodic boundary conditions, we set $U_0 = U_m$. Then, U_m can be removed from the discretization scheme and the finite difference derivatives of $U = (U_0, U_1, \dots, U_{m-1})^T$ at the endpoints are discretized as

$$\partial_r U_0 = \frac{h_{m-1}^2(U_1 - U_0) + h_0^2(U_0 - U_{m-1})}{h_{m-1}h_0(h_{m-1} + h_0)}, \tag{4.2.7}$$

$$\partial_r U_{m-1} = \frac{h_{m-2}^2(U_0 - U_{m-1}) + h_{m-1}^2(U_{m-1} - U_{m-2})}{h_{m-2}h_{m-1}(h_{m-2} + h_{m-1})}, \tag{4.2.8}$$

$$\partial_r^2 U_0 = \frac{2[h_{m-1}(U_1 - U_0) - h_0(U_0 - U_{m-1})]}{h_{m-1}h_0(h_{m-1} + h_0)} \tag{4.2.9}$$

$$\partial_r^2 U_{m-1} = \frac{2[h_{m-2}(U_0 - U_{m-1}) - h_{m-1}(U_{m-1} - U_{m-2})]}{h_{m-2}h_{m-1}(h_{m-2} + h_{m-1})} \tag{4.2.10}$$

where $h_i = r_{i+1} - r_i$. At the inner nodes $U_1, \ldots, U_{m-2}$, we use the centered difference approximations given by (4.1.6) and (4.1.7). Discretizing using the finite difference schemes and the backward Euler method gives a system of equations whose solution gives an approximation to the solution of (4.2.5).

with the above discretization will be called a "physical time solve", which we will label as a "P_t" solve moving forward. We note that this discretization arises from the centered difference approximation, but one could similarly employ an upwind approximation. Additionally, one could use an explicit time stepping scheme if desired such as fourth order Runge-Kutta.

The proposed iterative procedure for the advection-diffusion equation on a curve is given in Algorithm 3.

Algorithm 3: Single domain MP_t method on a curve

```
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
initialize the uniform mesh $r^{(0)}$;
Use the initial condition to obtain $U^{(0)}$;
for $k = 0, 1, \ldots,$ do
    Set $t^{(k+1)} = t^{(k)} + \Delta t$ ;
    Given the monitor function vector $F$ (which uses $U^{(k)}$), complete a mesh
      solve on the computational coordinate domain $\Omega = [0,1]$ with boundary
      values $r_\alpha = r(0)$ and $r_\alpha = r(1)$ to determine the new grid $r^{(k+1)}$. $F$ must
      be interpolated onto the new grid at each step of the equidistribution;
    Interpolate $U^{(k)}$ onto the new grid $r^{(k+1)}$;
    Complete a physical time solve on $r^{(k+1)}$ with periodic boundary
      conditions to obtain $U^{(k+1)}$;
end
```

In Algorithm 3, Δt is constant; this varies from the alternating procedure described in [31], where a variable time step size Δt is used. We remark that in Algorithm 3, we complete one mesh solve and one physical solve at each time step. There is no alternating loop inside of the time loop; this is a standard approach in the moving mesh literature. A disadvantage of this approach is that one can experience a mesh lag, meaning that the mesh starts to lag behind the physical solution. This issue could be fixed by doing additional mesh and physical solves in an alternating loop, where the alternating loop occurs for each time step. This approach would be more analagous to Algorithm 1 from Section 4.1.

We provide the solution of (4.2.5) with equidistribution using two different monitor functions in Figure 4.4. We see from Figure 4.4 that equidistributing by just function features, with monitor function (2.4.17), concentrates nodes only where the function needs it, compared to monitor function (2.4.19) which balances its concentration of nodes between both the curve and function features. For the monitor function (2.4.17), we do not best resolve the curve on the other areas on the ellipse. For example, we can see this visually in the second row of Figure 4.4 in the general region $r \in (0.5, 1)$, which corresponds to the regions $x \in (-3., 3)$ $y \in (-0.5, 0)$, and $u \approx 0$. We see that the monitor function (2.4.19) (right) is not resolving the curve as well in this region.

This demonstrates the importance of including both curve and function features in equidistribution, and also emphasizes the importance of visualizing the numerical results. The 2d plot of u versus r does not show the curve on which u is posed, making it seem as if a monitor function that only takes in function features will be the best choice.

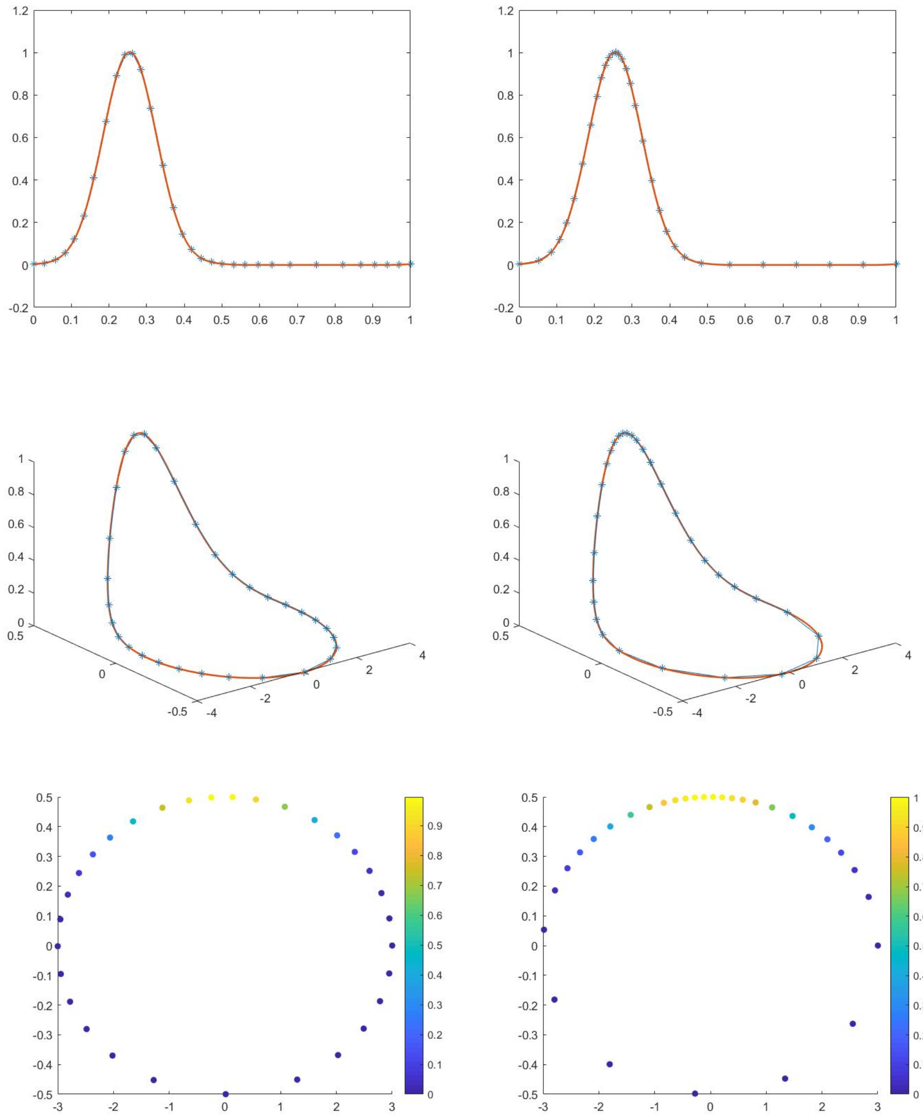


Figure 4.4: Solutions of (4.2.5), where the PDE is formed from the solution $u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}}$ with $\epsilon = 0.01$, posed on the ellipse (2.2.14) with radius $A = 3$ and $B = 0.5$. Solutions are determined from the single domain MP_t method on a curve and are given at $t_f = 0.05$, where $\Delta t = 0.0011$, giving $k = 45$ time steps of the algorithm. Solutions are given on a mesh of $m = 32$ nodes equidistributed with monitor function (2.4.19) (left) and on a mesh equidistributed with monitor function (2.4.17) (right). Top: 2d plot of discrete solution U versus r, plotted with the fine grid solution with $m_{fine} = 600$ nodes shown in solid red for comparison. Row 2: 3d plot of discrete solution U on $\mathbf{x}(r)$, plotted with the fine grid solution shown in solid red for comparison. Bottom: Color plot of discrete solution U on $\mathbf{x}(r)$.

As in Section 2.4, we begin with a coarse grid r and fine grid r_{fine} to determine the interpolation error. Since the function (4.2.6) is not exactly periodic, we will not determine interpolation errors compared to the "true" solution. Instead, we compare the numerical results to the discrete solution u_{fine}, determined on a fine uniform grid. Choosing the time t to evaluate the numerical solutions, the coarse grid r is equidistributed with a given monitor function, then the resulting $x = x(r), y = y(r)$ are linearly interpolated onto the fine grid r_{fine}, giving (x_{interp}, y_{interp}). The numerical solution at each time step U, found using backward Euler, is interpolated onto r_{fine}, giving u_{interp}. Then these interpolated values are compared to $x_{fine} = x(r_{fine}), y_{fine} = y(r_{fine})$, and u_{fine}. The error between the equidistributed grids and the numerical fine grid solution is then given by

$$e = \sqrt{(x_{fine} - x_{interp})^2 + (y_{fine} - y_{interp})^2 + (u_{fine} - u_{interp})^2}.$$

As in Section 2.4, we can also separate the error e into a "curve" and "function" error, where the curve error is defined as

$$e_c = \sqrt{(x_{fine} - x_{interp})^2 + (y_{fine} - y_{interp})^2},$$

and the function error is defined as

$$e_u = \sqrt{(u_{fine} - u_{interp})^2},$$

or $|u_{fine} - u_{interp}|$. We then take the maximum and Euclidean norms of the error vectors e, e_c, and e_u. Interpolation results are provided in Table 4.1.

From Table 4.1, we see that for (4.2.5) posed on the ellipse (2.2.14), we obtain lower errors than the uniform mesh in the maximum and Euclidean norms with the monitor functions (2.4.4) (both curve and function features) and (2.4.19) (both curve and function features). We remark that these interpolation errors are consistent with the discussion of the results in Figure 4.4; that is, equidistributing using both curve and function features generally produces lower overall interpolation errors than using only curve or function features.

Additionally, we look at the function error; that is, $||e_u||_\infty$ and $||e_u||_2$. We obtain lower function errors than the uniform mesh in the maximum and Euclidean norms with the monitor functions (2.4.4), (2.4.17) (just function features), and (2.4.19).

It is important to keep in mind that the most relevant errors are the overall errors $||e||_2$ and $||e||_\infty$, as they represent a combined error of both the curve and function. Additionally, while monitor functions (2.4.4) and (2.4.19) may seem to only slightly reduce the overall errors, the errors are reduced drastically in terms of percentage. For example, the monitor function (2.4.19) reduces the overall error $||e||_2$ by approximately 35% when compared to the fine uniform mesh. Additionally, as in Chapter 2, we remark that we are experimenting with sensible monitor functions rather than determining the optimal monitor function.

Table 4.1: Interpolation errors for the solution of (4.2.5), where the PDE is formed from the solution $u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}}$ with $\epsilon = 0.01$, posed on the ellipse (2.2.14) with radius $A = 3$ and $B = 0.5$. Interpolation errors are determined from the single domain MP_t method on a curve, where $t_f = 0.05$ and $\Delta t = 0.0011$, giving $k = 45$ time steps of the algorithm. The weights are chosen as $\omega = 0.1$ and $\omega_u = 0.01$ for all monitor functions. Here, we use $m = 30$ nodes and $M = 600$ nodes for the fine mesh used to compute the interpolation error.

Monitor Function	$\|\|e_c\|\|_\infty$	$\|\|e_c\|\|_2$	$\|\|e_u\|\|_\infty$	$\|\|e_u\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.4)	.0088153	0.004589	0.042295	0.0057645	0.042297	.0073681
(2.4.17)	0.069733	0.035186	0.0050103	0.0013508	0.069734	0.035212
(2.4.3)	.008226	0.004234	0.068855	0.0092235	0.068861	0.010149
(2.4.5)	0.01486	0.0059667	0.1795	0.031676	0.17982	0.032233
(2.4.6)	0.010366	0.0047088	0.18993	0.028953	0.1901	0.029334
(2.4.19)	0.013927	0.0051619	0.020762	0.0033246	0.020798	.0061399
uniform mesh	.0082116	0.0042607	0.064116	0.0084225	0.064121	.0094388

As another example, we choose the same PDE (4.2.5) with a solution of $u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}}$, with $\epsilon = 0.001$. Reducing the value of ϵ will produce a steeper "pulse" in the function. This means that there will generally be a greater concentration of nodes in that area of the pulse than in Figure 4.4, as the function is increasing and decreasing more rapidly in that region. We provide the solution with equidistribution using two different monitor functions in Figure 4.5. As in Figure 4.4, we see from Figure 4.5 that equidistributing by just function features with monitor function (2.4.17) concentrates nodes only where the function needs it, compared to monitor function (2.4.6) which balances its concentration of nodes between both the curve and function features.

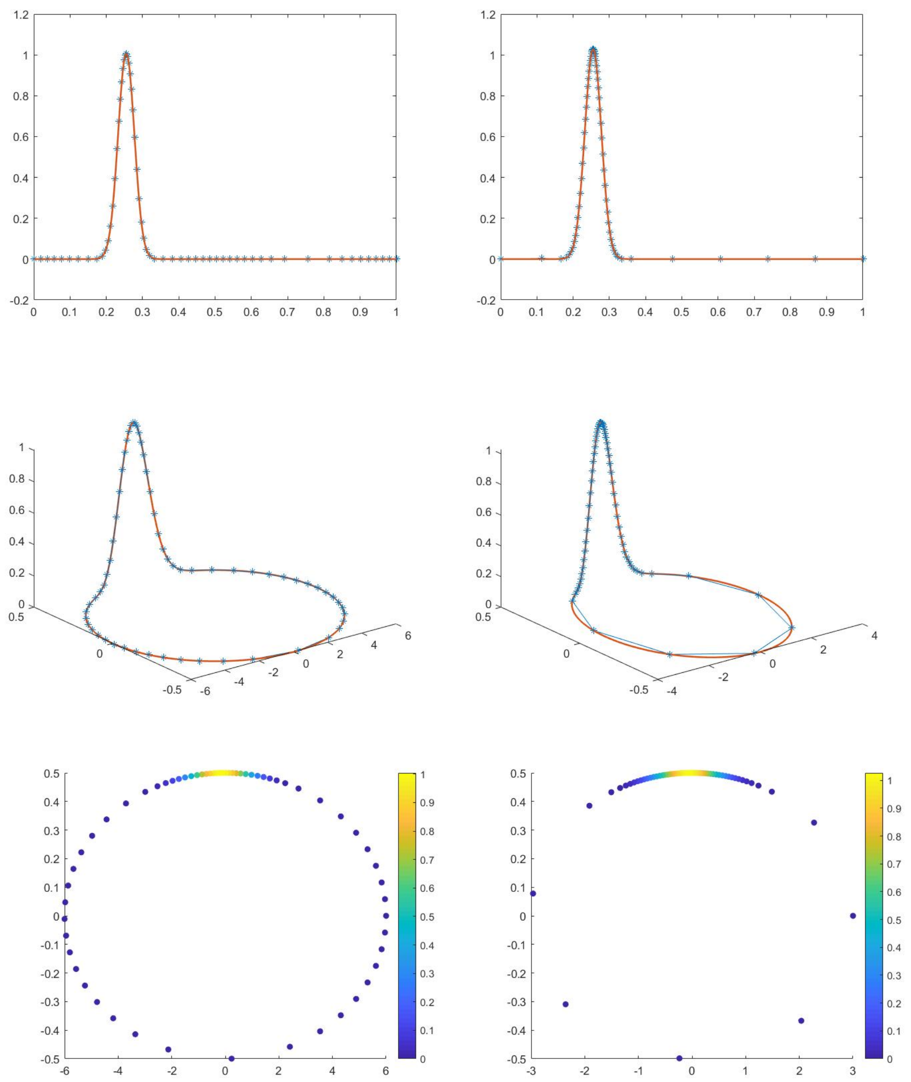


Figure 4.5: Solutions of (4.2.5), where the PDE is formed from the solution $u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}}$ with $\epsilon = 0.001$, posed on the ellipse (2.2.14) with radius $A = 3$ and $B = 0.5$. Solutions determined from the single domain MP_t method on a curve are given at $t_f = 0.05$, where $\Delta t = 0.00028$, giving $k = 180$ time steps of the algorithm. Solutions are given on a mesh of $m = 60$ nodes equidistributed with monitor function (2.4.19) (left) and on a mesh equidistributed with monitor function (2.4.17) (right). Top: 2d plot of discrete solution U versus r, plotted with the fine grid solution with $m_{fine} = 600$ nodes shown in solid red for comparison. Row 2: 3d plot of discrete solution U on $\mathbf{x}(r)$, plotted with the fine grid solution shown in solid red for comparison. Bottom: Color plot of discrete solution U on $\mathbf{x}(r)$.

Interpolation error results are shown in Table 4.2. From Table 4.2, we see that when (4.2.5) is solved on the ellipse (2.2.14), we obtain lower errors than the uniform mesh in the maximum and Euclidean norms with the monitor functions (2.4.4) (both curve and function features) and (2.4.19) (both curve and function features).

Additionally, looking specifically at the function error $||e_u||$, we obtain lower function errors than the uniform mesh in the maximum and Euclidean norms with the monitor functions (2.4.4), (2.4.17) (just function features), and (2.4.19). However, we once again remark that the overall errors $||e||_2$ and $||e||_\infty$ are the most relevant, as they consider both the curve and function error.

This is the same result as Table 4.1. Generally, we remark that decreasing the value of ϵ will produce a greater need for equidistribution (specifically equidistributing with function features), given that the "pulse" is steeper and will need a greater concentration of nodes in that area.

Table 4.2: Interpolation errors for the solution of (4.2.5), where the PDE is formed from the solution $u(r,t) = e^{-\frac{(r-\frac{1}{4}-\frac{1}{2}t)^2}{\epsilon}}$ with $\epsilon = 0.001$, posed on the ellipse (2.2.14) with radius $A = 3$ and $B = 0.5$. Interpolation errors are determined from the single domain MP_t method on a curve, where $t_f = 0.01$ and $\Delta t = 0.00028$, giving $k = 36$ time steps of the algorithm. The weights are chosen as $\omega = 0.1$ and $\omega_u = 0.05$ for all monitor functions. Here, we use $m = 60$ nodes and $M = 600$ nodes for the fine mesh used to compute the interpolation error.

Monitor Function	$\|\|e_c\|\|_\infty$	$\|\|e_c\|\|_2$	$\|\|e_u\|\|_\infty$	$\|\|e_u\|\|_2$	$\|\|e\|\|_\infty$	$\|\|e\|\|_2$
(2.4.4)	0.018295	0.0093129	0.027116	0.0056112	0.027286	0.010873
(2.4.17)	0.070478	0.02839	0.0058895	0.0022566	0.070497	0.028479
(2.4.3)	0.015639	0.0083006	0.030311	0.0067759	0.030464	0.010715
(2.4.5)	0.017509	0.0090106	0.13872	0.028206	0.13951	0.02961
(2.4.6)	0.014799	0.0080526	0.10247	0.019733	0.10303	0.021312
(2.4.19)	0.021088	0.0091877	0.016009	0.0043339	0.021088	0.010159
uniform mesh	0.016347	0.0086033	0.027228	0.0061244	0.027365	0.010561

Chapter 5

The Multi-domain Equidistribution for the Solutions of Differential Equations on Static Curves

In this chapter, we provide and study Classical Schwarz variants for the single domain algorithms provided in Chapter 4, in order to solve multi-domain problems. To do this, we employ an alternating technique, alternating between a mesh PDE and physical PDE. In the static boundary layer case, we begin with a uniform mesh and alternate at each iteration until a tolerance is reached between approximations. In the time-dependent PDE case, we begin with the initial condition and alternate at each time step until the desired final time t_f.

In this chapter, we set up DD iterations in the r variable to solve the physical PDE to obtain a numerical approximation to the solution $u(r)$. Another approach would be to set up the DD iteration in the computational ξ coordinate. This would involve partitioning ξ into fixed, uniform subdomains, and considering the solution as $u(r(\xi))$, transforming the PDE by the chain rule. This is discussed in further detail in Chapter 6.

We remark that the methods described in this chapter could be extended to optimized Schwarz iterations. One would simply change the Dirichlet transmission conditions to the Robin transmission conditions discussed in Section 3.1.2.

5.1 The Multi-Domain Solution of Static Boundary Layer Problems

We consider an alternating approach, first solving the mesh BVP given by (2.2.2) in parallel before solving the boundary layer problem given by (4.1.11) in parallel, where (2.2.2) is given on the interval $\xi \in [0,1]$ by

$$(F(r)r_\xi)_\xi = 0, \qquad r(0) = 0, \;\; r(1) = b, \tag{5.1.1}$$

and (4.1.5) is given on the interval $r \in [0,b]$ by

$$-\epsilon\psi\left(\psi_r u_r + \psi u_{rr}\right) + k\psi u_r + cu = f, \qquad u(0) = \gamma_0, \;\; u(b) = \gamma_b. \tag{5.1.2}$$

We proceed by setting up the DD iteration for the mesh $r(\xi)$ in the ξ variable and setting up the DD iteration for the physical solution in the r variable.

Consider a uniform computational grid $\xi \in [0,1]$ broken into S overlapping subdomains such that $\Omega_i = \xi_i = [\alpha_i, \beta_i]$, $i = 1,\dots,S$, where $\alpha_1 = 0$ and $\beta_S = 1$. Additionally, consider the initial meshes, r_i, given by

$$\begin{aligned} r_i &= (r_i(\alpha_i),\dots,r_i(\beta_i))^T, \\ &= (r_{i,0}, r_{i,1},\dots,r_{i,m-1}, r_{i,m})^T, \end{aligned} \tag{5.1.3}$$

$i = 1,\dots,S$, where $r_{1,0} = 0$ and $r_{s,m} = b$, where r_i are uniformly spaced. We begin with an initial guess $u^{(0)}$ and choose $U_i^{(0)} = u^{(0)}(r_i)$. We will complete a parallel mesh solve to obtain $r_i^{(n)}$ on ξ_i, and use this mesh to compute the physical numerical solution $U_i^{(n)}$ on $r_i^{(n)}$.

We require an initial guess $u^{(0)}$ in order to provide the Dirichlet boundary conditions needed for each subdomain. The most straightforward way to approach this problem is to alternate between complete DD solves of the physical and mesh equations until the sequential iterates converge within a given tolerance. In this method, described in Algorithm 4, the outermost loop is the alternating mesh and physical PDE iteration with index n. At each alternating iteration, the DD mesh and physical solves are executed until convergence. The mesh and physical solves are continued until the mesh and physical solves cease to change from the previous approximation. This gives us both a final result for our mesh, and a final result for our solution.

Algorithm 4: $(M_\infty P_\infty)_\nu$ Method

```
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
Choose a uniform initial mesh $r^{(0)}$;
Initialize initial guess $U^{(0)}$;
for $n = 1, 2, \ldots,$ do
    initialize $r_i^{(0)} = r_i^{(n-1)}$ for $i = 1, \ldots, S$ ;
    for $c = 1, 2, \ldots,$ do
        for $i = 1, \ldots, S$ do
            Complete a mesh solve on the computational coordinate domain
             $\Omega_i = [\alpha_i, \beta_i]$ with boundary values
             $\begin{cases} r_\alpha = 0, & r_\beta = r_{i+1}^{(c-1)}(\beta_i), & \text{for } i = 1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = r_{i+1}^{(c-1)}(\beta_i), & \text{for } i = 2, \ldots, S-1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = b, & \text{for } i = S, \end{cases}$
            to obtain $r_i^{(c)}$ ;
        end
        if $\max_i(||r_i^{(c)} - r_i^{(c-1)}||_\infty) < tol/10$ then
            BREAK LOOP;
        end
    end
    Result: $r_i^{(n)}$ for $i = 1, \ldots, S$
    for $c = 1, 2, \ldots,$ do
        for $i = 1, \ldots, S$ do
            Complete a physical solve on $\Omega = r_i^{(n)}$, with boundary values
             $\begin{cases} \mathcal{B}_1 = \gamma_0, & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) & \text{for } i = 1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) & \text{for } i = 2, \ldots, S-1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = \gamma_b, & \text{for } i = S, \end{cases}$
            to obtain $U_i^{(c)}$ on $\Omega_i$ ;
        end
        if $\max_i(||U_i^{(c)} - U_i^{(c-1)}||_\infty) < tol/10$ then
            BREAK LOOP;
        end
    end
    Result: $U_i^{(n)}$ for $i = 1, \ldots, S$
    if $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < tol$ then
        BREAK LOOP;
    end
end
```

We refer to this method as the $(M_\infty P_\infty)_\nu$ method. Here, the notation M_∞ and P_∞ denote a complete mesh solve and physical solve, respectively. Here, "complete" means the DD solves are continued until convergence. The subscript $(\cdot)_\nu$ indicates that ν alternating iterations are needed for convergence within a given tolerance. We note that the ν subscript could also be written as ∞, as in practice we run alternating mesh and physical iterations until a tolerance is achieved between two consecutive approximations. However, we keep the subscript ν to be consistent. We note that there are many moving pieces in the $(M_\infty P_\infty)_\nu$ method. Depending on how we approach the alternating method, we can lose accuracy in our solutions.

We provide numerical results for the example used in Section 4.1 given by

$$\epsilon\psi\left(\psi_r u_r + \psi u_{rr}\right) + \psi u_r = 1, \quad u \text{ on } \mathcal{C}, \quad u(0) = 1, \;\; u(1) = 1. \tag{5.1.4}$$

A plot showing the error between the solution of the $(M_\infty P_\infty)_\nu$ algorithm for $S = 2$ subdomains and the corresponding single domain solution as a function of m is given in Figure 5.1. The tolerance is chosen to be 10^{-8}. The error $||e||_\infty$ refers to the maximum norm of the error between the discrete solution of the $(M_\infty P_\infty)_\nu$ method and the discrete solution of the single domain MP method.

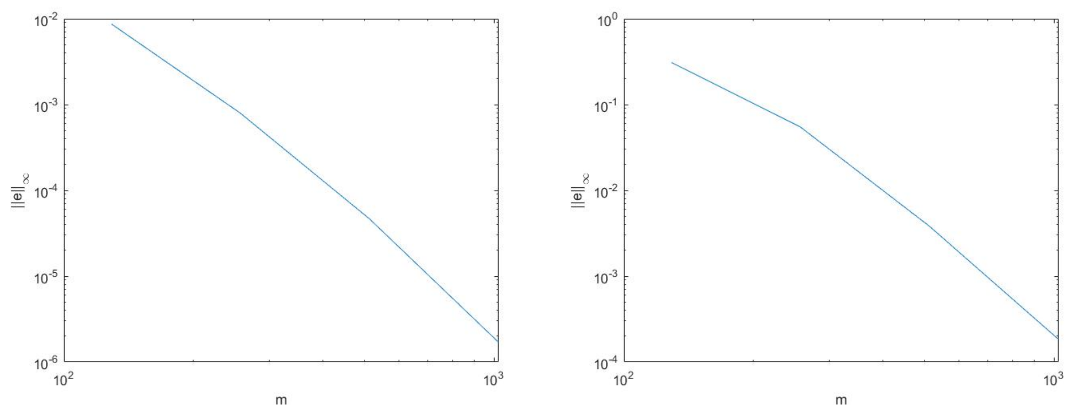


Figure 5.1: A plot of the infinity norm of the difference between the $(M_\infty P_\infty)_\nu$ solution and the single domain discrete solution used to solve (5.1.4) for a varying number of nodes, m. Errors are given for the mesh $r^{(n)}$ (left) and physical solution $U^{(n)}$ (right). Here, the monitor function (2.4.6) is used.

It is clear that with a low DD tolerance such as 10^{-10} and high number of nodes $m = 10^3$, we are still obtaining a relatively large error between the single domain solution and the $(M_\infty P_\infty)_\nu$ solution. This appears to be due to the interpolation at each step of the equidistribution in the $(M_\infty P_\infty)_\nu$ method. Interpolation, while a necessary step when an analytical solution is unknown, produces an interpolation error that will have an effect on the accuracy of our solution. This error results in an immediate error (from the single domain solution) in the $(M_\infty P_\infty)_\nu$ method that occurs within the first DD iteration and maintains the same order of magnitude throughout the entire iteration. The error from the single domain mesh given in Figure 5.1 is determined to be $O(h^4)$, where $h = \max_i h_i$ and $h_i = r_i - r_{i-1}$. This is also the error which generally results from the cubic spline interpolation used in our algorithm. This was determined by the slope of a loglog plot of $||e||_\infty$ versus h. The error from the single domain physical solution given in Figure 5.1 is determined to be $O(h^{3.6})$.

The $O(h^4)$ error from the single domain solution is slightly unexpected as the method of taking the required derivative of a cubic spline by taking the first and second derivative of each piecewise cubic generally produces $O(h^3)$ and $O(h^2)$ errors, respectively. This is shown in Figure 5.2 for the function $u(x) = \sin(x)$.

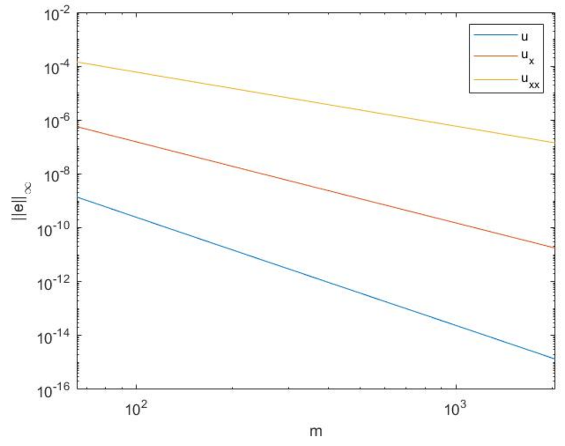


Figure 5.2: A plot showing the interpolation errors resulting from cubic splines. The interpolation errors are shown for the function $u(x) = \sin(x)$ and its corresponding first and second derivatives.

From Remark 4.1.1, we will also be taking the derivative of a interpolated vector to obtain F' from F at each Newton iteration of the mesh solve. Additionally, after every complete DD physical solve, we must interpolate the physical solution $U_i^{(n-1)}$ from the mesh at the previous time step onto the initial guess used in each mesh solve.

Therefore, we will also be taking the derivative of an interpolated vector to obtain the first or second derivatives of the physical solution, which are used to determine $F(r, u')$ or $F(r, u'')$ at the beginning of each mesh solve.

We note that in Figure 5.1, mesh smoothing was not used. While useful in practice, mesh smoothing will have an effect on the order of convergence; this is shown in Figure 5.3. In Figure 5.3, smoothing is used in both the single domain and DD methods. Smoothing the monitor function in the $(M_\infty P_\infty)_\nu$ method increases the error to $O(h^2)$ from the single domain mesh, where the single domain mesh is also smoothed with identical smoothing parameters.

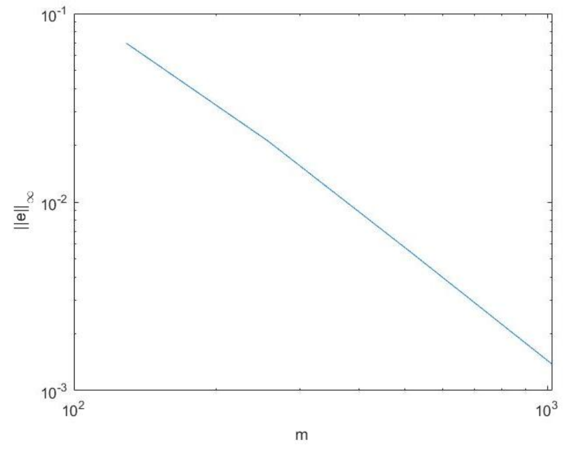


Figure 5.3: A plot of the infinity norm of the difference between the $(M_\infty P_\infty)_\nu$ solution and the single domain solution used to solve (5.1.4) for a varying number of nodes, m. The error is given for the mesh, comparing $r_i^{(n)}$ for $i = 1, 2$ to the single domain solution $r^{(n)}$. Here, mesh smoothing is used in the equidistribution step. Here, we use monitor function (2.4.6)

To demonstrate further that the interpolation error is causing this discrepancy, we also include Figure 5.4 showing the error from the single domain solution in the case where u is a known analytical function, rather than a discrete approximation to a differential equation solution. In this case, $F(r)$ can be exactly evaluated at each step of the equidistribution, and no approximation/interpolation is needed. Due to this lack of interpolation, the DD variant converges to the single domain solution within the DD tolerance. In this case, the algorithm used will be called the $(M_\infty)_\nu$ method, as there is no physical solve.

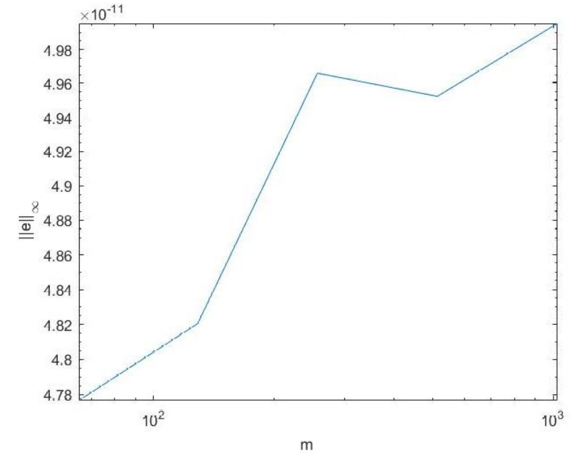


Figure 5.4: A plot of the infinity norm of the difference between the DD and the single domain solution for the equidistribution of a known function $u = \cos(2\pi r)$ on the ellipse (2.2.14) for a varying number of nodes, m. The error is given for the mesh, comparing $r_i^{(n)}$ for $i = 1, 2$ to the single domain solution $r^{(n)}$. Here, the monitor function (2.4.6) is used with a DD tolerance of 10^{-10}.

From theory, we expect the cubic spline to produce a smaller interpolation error than linear interpolation; this is expected to be $O(h^4)$ versus $O(h^2)$, respectively. If F was interpolated using linear interpolation, we would calculate F' by taking the derivative of each piecewise linear equation. The method of taking the first derivative of a piecewise linear interpolant by taking the first derivative of each piecewise linear generally produces $O(h)$ error. This is shown in Figure 5.5 for the function $u(x) = \sin(x)$.

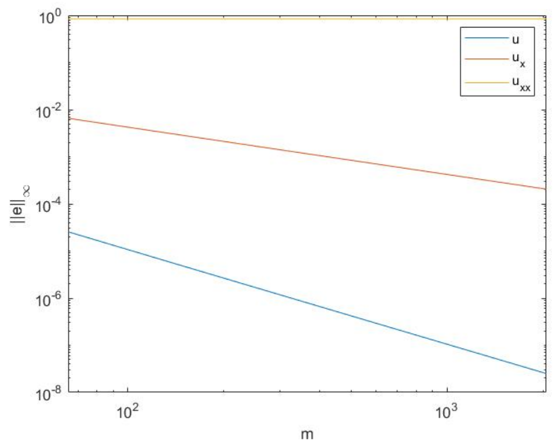


Figure 5.5: A plot showing the interpolation errors resulting from piecewise linear interpolation. The interpolation errors are shown for the function $u(x) = \sin(x)$ and its corresponding first and second derivatives.

We see from Figure 5.5 that this is not a method we can employ to obtain u'' for a numerical interpolant for u. Since a linear equation has a second derivative equal to zero, we will only be able to determine first derivatives using this method. This means that if we are using a monitor function such as (2.4.19) which uses u'', we can calculate u'' by the finite difference approximation (4.1.7), which we have stated gives $O(h^2)$ error on a uniform mesh and $O(h)$ error on a non-uniform mesh.

In practice, however, we are also not achieving convergence to the single domain when using linear interpolation and determining F' or u' by interpolating the piecewise linear equations. In the case of using linear interpolation, we are only obtaining convergence to the single domain solution when computing F and u' by the finite difference approximation (4.1.6).

As expected, using piecewise linear interpolation increases the error, giving approximately $O(h^{1.6})$ error from the single domain discrete solution for the mesh, and $O(h)$ error from the single domain discrete solution for the physical solution. This is shown in Figure 5.6.

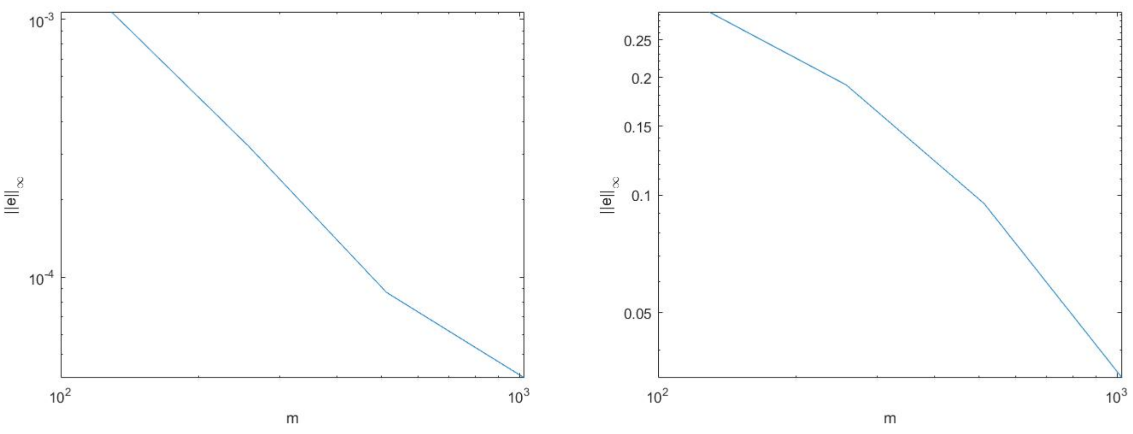


Figure 5.6: A plot of the infinity norm of the difference between the $(M_\infty P_\infty)_\nu$ solution and the single domain discrete solution used to solve (5.1.4) for a varying number of nodes, m. Here, linear interpolation was used. DD errors are given for the mesh $r^{(n)}$ (left) and physical solution $U^{(n)}$ (right). Here, we use monitor function (2.4.6).

We conclude that the benefits of using a differentiable cubic spline interpolation generally outweigh the extra computations required for the splines compared to linear interpolation.

A final note about the $(M_\infty P_\infty)_\nu$ method and Figure 5.1 is that we do not obtain DD convergence for the mesh for a small number of nodes m. Specifically, the Newton solve fails to converge on the subdomains. There are a number of factors that may be causing this. From DD theory, we expect the DD mesh solve to converge if m is sufficiently large; this was proven in the discrete case on the line in [26]. In the boundary layer case, we also note that for small m, u will be jagged and unsmooth, containing “wiggles” as the boundary layer is attempted to be resolved. Therefore, it is likely the case that for small m (and even smaller subdomain sizes), there are simply not enough nodes to compute the solution. A potential fix would be to complete our physical solves using an upwind method instead of a centered difference method, therefore suppressing the wiggles in the discrete solution. We could also discretize using the upwind method in early physical iterates, then switch to centered difference methods once the boundary layer has been more resolved. We could also address this problem using a continuation in ϵ (where ϵ is the constant given by the original boundary layer problem). This would involve beginning with a larger value of ϵ, which would result in fewer “wiggles”. Then, we use this solution to systematically reduce ϵ until the original BVP is solved. Additionally, we could proceed with smoothing,

smoothing the interpolant u to remove the wiggles.

There are multiple ways to approach the solution of this boundary layer problem with domain decomposition. A slight variant would be to complete either a fixed number C of mesh and physical PDE solves at each DD iteration. That is, we use "inexact" solves at each alternating iteration. This method is described in Algorithm 5.

We refer to this method as the $(M_C P_C)_\nu$ method, as C mesh solves and C physical solves are computed at each alternating iteration n, where C is fixed. This method also produces $O(h^4)$ error from the single domain solution; this is shown in Figure 5.7. We note that we obtain $O(h^4)$ convergence for all $C > 1$.

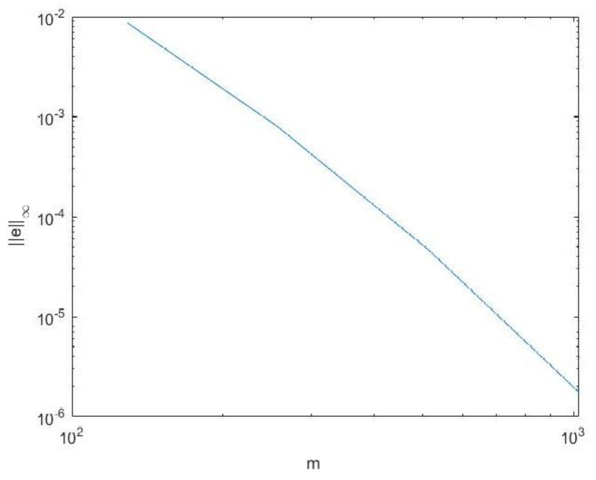


Figure 5.7: A plot of the infinity norm of the error between the $(M_C P_C)_\nu$ solution with $C = 4$ and single domain solution used to solve (5.1.4) for a varying number of nodes, m. Here, we use monitor function (2.4.6).

To further show that the choice of C has little effect on the convergence, we provide a plot of DD error versus C for the $(M_C P_C)_\nu$ method. Here, the DD error is given from the discrete single domain solution obtained from the single domain MP method in Figure 5.8. We see that while error is slightly reduced as C is increased, there is little to no C dependence on error, as the slope of the line on a semi-log scale is ≈ 0.

Algorithm 5: $(M_C P_C)_\nu$ Method

```
Result:
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
Choose a uniform initial mesh $r^{(0)}$;
Initialize initial guess $U^{(0)}$;
for $n = 1, 2, \dots,$ do
    initialize $r_i^{(0)} = r_i^{(n-1)}$ for $i = 1, \dots, S$ ;
    for $c = 1, \dots, C$ do
        for $i = 1, \dots, S$ do
            Complete a mesh solve on the computational coordinate domain
             $\Omega_i = [\alpha_i, \beta_i]$ with boundary values
            $\begin{cases} r_\alpha = 0, & r_\beta = r_{i+1}^{(c-1)}(\beta_i), \quad \text{for } i = 1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = r_{i+1}^{(c-1)}(\beta_i), \quad \text{for } i = 2, \dots, S-1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = b, \quad \text{for } i = S, \end{cases}$
            to obtain $r_i^{(c)}$ ;
        end
    end
    Result: $r_i^{(n)}$ for $i = 1, \dots, S$
    for $c = 1, \dots, C$ do
        for $i = 1, \dots, S$ do
            Complete a physical solve on $\Omega = r_i^{(n)}$, with boundary values
            $\begin{cases} \mathcal{B}_1 = \gamma_0, & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) \quad \text{for } i = 1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) \quad \text{for } i = 2, \dots, S-1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = \gamma_b, \quad \text{for } i = S, \end{cases}$
            to obtain $U_i^{(c)}$ on $\Omega_i$ ;
        end
    end
    Result: $U_i^{(n)}$ for $i = 1, \dots, S$
    if $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < tol$ then
        BREAK LOOP;
    end
end
```

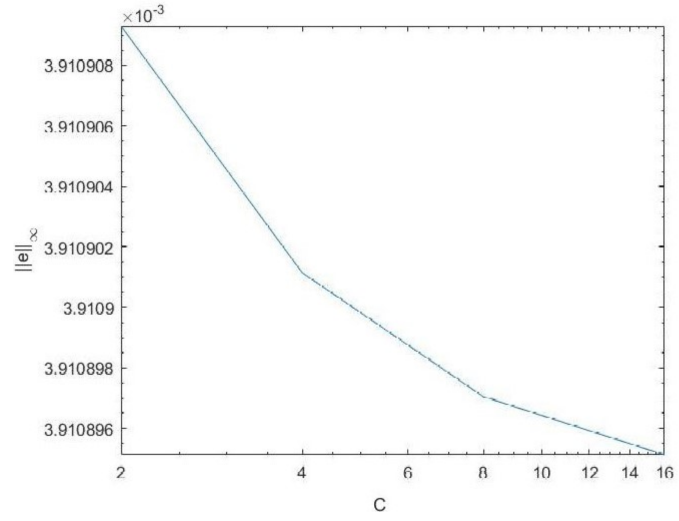


Figure 5.8: A plot of the infinity norm of the error between the $(M_C P_C)_\nu$ solution with various C values and the single domain solution used to solve (5.1.4). Here, we equidistribute $m = 512$ nodes with monitor function (2.4.6).

Remark 5.1.1. *In some cases, it is numerically necessary to compute the nonlinear solves using a damping iteration. The damping iteration is given by*

$$r_{n+1} = r_n + \frac{1}{d}\delta, \tag{5.1.5}$$

where $d \geq 1$ is the damping parameter and

$$J\delta = -G, \tag{5.1.6}$$

is a Newton update given by 1 linear solve at each iteration. Here, J represents the Jacobian of G, where $G((r_0, r_1, \ldots, r_m)^T) = 0$ results from the discretization of the nonlinear equations. In this case, as we are interpolating an unknown discrete u, we must keep all values of r within the original interval $[0, b]$ to ensure that no extrapolation is occurring; we also must keep r monotone at each Newton step. We build these conditions into the damped Newton iteration, doubling the damping parameter until all conditions are met.

Table 5.1 provides insight into the efficiency of the $(M_\infty P_\infty)_\nu$ method and the $(M_C P_C)_\nu$ method. Table 5.1 includes the number of linear solves used to obtain convergence with each method. As linear solves will take up the large majority of computation time, this is considered an adequate way to measure the approximate computation time.

Table 5.1: The number of linear solves for the $(M_C P_C)_\nu$ and $(M_\infty P_\infty)_\nu$ methods to solve the boundary layer problem (4.1.11) with $m = 256$ nodes. The value of ϵ in (4.1.11) is varied. Here, $n = 2$ subdomains are used, the overlap is $O = 2$ nodes, and $C = 4$. All DD methods converge to the same solution. All methods were stopped when $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < 10^{-8}$.

ϵ	$(M_C P_C)_\nu$ linear solves	$(M_\infty P_\infty)_\nu$ linear solves
0.1	7650	14661
0.05	9400	19779
0.01	18698	71017
0.005	37024	628328

The results of Table 5.1 show that as we decrease ϵ (and therefore increase the need for equidistribution), the $(M_C P_C)_\nu$ method becomes more efficient compared to the $(M_\infty P_\infty)_\nu$ method.

It is clear that the $(M_C P_C)_\nu$ method allows for variation in the number of partial solves per iteration. We fix $\epsilon = 0.01$ in the boundary layer problem (5.1.4) and explore the results as we increase or decrease C. Results are shown in Table 5.2. We note that for $C > 1$ in this method, we are obtaining $O(h^4)$ error from the single domain solution.

Table 5.2: The number of linear solves for the $(M_C P_C)_\nu$ method with various C values to solve the boundary layer problem (5.1.4) with $m = 256$ nodes. Here, $n = 2$ subdomains are used, the overlap is $O = 2$ nodes, and $\epsilon = 0.01$. All methods were stopped when $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < 10^{-8}$.

C	$(M_C P_C)_\nu$ method linear solves
2	8546
4	8848
8	9279
16	9439
32	9589

From Table 5.2, we see that, as expected, the global number of linear solves increases as we increase C. However, choosing $C = 1$ does not result in convergence.

Another way to approach this solution procedure would be to compute mesh and PDE solves until tolerances tol_p and tol_m are reached. Generally, we choose $tol_p, tol_m > tol$ to speed up convergence. This is described in Algorithm 6.

We refer to this method as the $(M_{tol_m} P_{tol_p})_\nu$ method, as each mesh and physical solve are completed to a specific tolerance. In practice, the $(M_{tol_m} P_{tol_p})_\nu$ method proved to be non-robust; setting a larger tolerance within the inner DD loop restricts the overall convergence of the method. For example, we find that if the inner DD tolerance is set to be larger than the outer alternating tolerance such that $tol_m, tol_p > tol$, the alternating loop will only converge with a tolerance of $\max(tol_m, tol_p)$. This causes the outer alternating loop n to not reach its stopping criteria of $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < tol$, as $tol < tol_m, tol_p$.

We omit numerical results for this method and recommend against it, except for the situation where tol_m and tol_p are chosen based on the outer loop. That is, after each alternating iteration, we chose tol_m and tol_p such that they are both less than the error of the outer alternating iteration. So at each alternating iteration n for subdomain i, we choose $tol_m^{(n)}, tol_p^{(n)} < \max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty)$. This choosing of inner tolerances such that they are ensured to be smaller than the outer tolerances is common practice when completing inexact solves.

5.2 The Multi-domain Solution of Time-dependent Differential Equations on Static Curves

The most natural approach to solve time-dependent PDEs in parallel comes from Cai [5, 6] (described on an interval $[a, b]$). In this method, we discretize the PDE in time and solve the resulting sequence of elliptic problems with domain decomposition. We adapt this approach to time-dependent PDEs posed on curves.

To form the elliptic problem, we discretize the PDE in time using an implicit method. For the general linear time-dependent PDE $u_t = \mathcal{L}u, x \in \Omega$, discretizing with backward Euler, for each k, we obtain the elliptic problems

$$u^{(k+1)} = u^{(k)} + \Delta t \mathcal{L} u^{(k+1)}, \quad x \in \Omega, \tag{5.2.1}$$

for each time step $k = 1, 2, \ldots, n$. On two subdomains, the parallel iteration is given by completing DD iterations to convergence at each time step k. The parallel DD

Algorithm 6: $(M_{tol_m} P_{tol_p})_\nu$ Method

```
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
Choose a uniform initial mesh $r^{(0)}$;
Initialize initial guess $U^{(0)}$;
for $n = 1, 2, \ldots,$ do
    initialize $r_i^{(0)} = r_i^{(n-1)}$ for $i = 1, \ldots, S$ ;
    for $c = 1, 2, \ldots,$ do
        for $i = 1, \ldots, S$ do
            Complete a mesh solve on the computational coordinate domain
             $\Omega_i = [\alpha_i, \beta_i]$ with boundary values
            $\begin{cases} r_\alpha = 0, & r_\beta = r_{i+1}^{(c-1)}(\beta_i), \quad \text{for } i = 1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = r_{i+1}^{(c-1)}(\beta_i), \quad \text{for } i = 2, \ldots, S-1, \\ r_\alpha = r_{i-1}^{(c-1)}(\alpha_i), & r_\beta = b, \quad \text{for } i = S, \end{cases}$
            to obtain $r_i^{(c)}$ ;
        end
        if $\max_i ||r_i^{(c)} - r_i^{(c-1)}||_\infty < tol_m$ then
            BREAK LOOP;
        end
    end
    Result: $r_i^{(n)}$
    for $c = 1, 2, \ldots,$ do
        for $i = 1, \ldots, S$ do
            Complete a physical solve on $\Omega = r_i^{(n)}$, with boundary values
            $\begin{cases} \mathcal{B}_1 = \gamma_0, & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) \quad \text{for } i = 1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = U_{i+1}^{(c-1)}(r_{i+1}^{(n)}(\beta_i)) \quad \text{for } i = 2, \ldots, S-1, \\ \mathcal{B}_1 = U_{i-1}^{(c-1)}(r_{i-1}^{(n)}(\alpha_i)), & \mathcal{B}_2 = \gamma_b, \quad \text{for } i = S, \end{cases}$
            to obtain $U_i^{(c)}$ on $\Omega_i$ ;
        end
        if $\max_i ||U_i^{(c)} - U_i^{(c-1)}||_\infty < tol_p$ then
            BREAK LOOP;
        end
    end
    Result: $U_i^{(n)}$
    if $\max_i(||U_i^{(n)} - U_i^{(n-1)}||_\infty, ||r_i^{(n)} - r_i^{(n-1)}||_\infty) < tol$ then
        BREAK LOOP;
    end
end
```

iteration at time step k is given as follows: for $n = 0, 1, \ldots$, solve

$$\begin{aligned} u_1^{(n)} &= u^{(k-1)} + \Delta t \mathcal{L} u_1^{(n)}, \quad x \in \Omega, \quad \mathcal{B}_1 u_1^{(n)} = \mathcal{B}_1 u_2^{(n-1)}, \\ u_2^{(n)} &= u^{(k-1)} + \Delta t \mathcal{L} u_2^{(n)}, \quad x \in \Omega, \quad \mathcal{B}_2 u_2^{(n)} = \mathcal{B}_2 u_1^{(n-1)}, \end{aligned} \tag{5.2.2}$$

with given transmission and boundary conditions. Here, $u^{(k-1)}$ is fixed at each time step and becomes a source term for time step k. We can view the time loop as the "outer" loop and the DD iteration as the "inner" loop. Here, we use Dirichlet transmission conditions, giving a classical Schwarz method.

Consider the example problem proposed in Section 4.2; that is, the advection diffusion equation on a closed curve with periodic boundary conditions. To obey the periodic boundary conditions $u(0) = u(b)$ on $\Omega = [0, b]$ with domain decomposition, we refer to the method described by Qaddouri et. al [46]. Following the authors of [46], we partition the domain $\Omega = [0, b]$ into two equally sized subdomains $\Omega_1 = r_1 = [0, \frac{b}{2} + \delta]$, $\Omega_2 = r_2 = [\frac{b}{2}, b + \delta]$. While this may seem like an odd partitioning due to $r = b + \delta$ being outside of the domain $[0, b]$, it is necessary to "wrap" around the periodic domain such that the subdomains overlap at the endpoints of the domain $[0, b]$. If the solution is periodic on $[0, b]$ such that $u(0) = u(b)$, then $u(\delta) = u(b + \delta)$ also holds, and so on.

For a general static differential equation $\mathcal{L}u(r) = f(r)$ with periodic boundary conditions $u(0) = u(b)$, the parallel iteration is as follows: for $n = 0, 1, \ldots$, solve

$$\begin{aligned} \mathcal{L} u_1^{(n)} &= f(r), r \in \Omega_1, & \mathcal{L} u_2^{(n)} &= f(r), r \in \Omega_2, \\ u_1^{(n)}(0) &= u_2^{(n-1)}(b), & u_2^{(n)}\left(\frac{b}{2}\right) &= u_1^{(n-1)}\left(\frac{b}{2}\right), \\ u_1^{(n)}\left(\frac{b}{2} + \delta\right) &= u_2^{(n-1)}\left(\frac{b}{2} + \delta\right) & u_2^{(n)}(b + \delta) &= u_1^{(n-1)}(\delta). \end{aligned} \tag{5.2.3}$$

Here, the iteration (5.2.3) obeys the periodic boundary conditions of the physical problem while still having Dirichlet transmission conditions on both subdomains as required by classical Schwarz. The authors of [46] also provide an optimized Schwarz iteration for the Helmholtz equation.

Combining the methods of solving a sequence of elliptic problems to solve time-dependent PDEs and the DD iteration above for the periodic problem, we obtain the iteration at time step k as follows: for $n = 0, 1, \ldots,$ solve

$$\begin{aligned} u_1^{(n)} &= u^{(k-1)} + \Delta t \mathcal{L} u_1^{(n)}, \quad r \in \Omega_1, & u_2^{(n)} &= u^{(k-1)} + \Delta t \mathcal{L} u_2^{(n)}, \quad r \in \Omega_2, \\ u_1^{(n)}(a) &= u_2^{(n-1)}(b), & u_2^{(n)}\left(\frac{b}{2}\right) &= u_1^{(n-1)}\left(\frac{b}{2}\right), \\ u_1^{(n)}\left(\frac{b}{2}+\delta\right) &= u_2^{(n-1)}\left(\frac{b}{2}+\delta\right) & u_2^{(n)}(b+\delta) &= u_1^{(n-1)}(\delta). \end{aligned} \tag{5.2.4}$$

While (5.2.4) is a reasonable parallel iteration, we wish to add in the layer of equidistribution in an alternating method similar to the single domain iteration given in Algorithm 3. Combining the methods of solving a sequence of elliptic problems to solve time-dependent PDEs using DD, implementing periodic boundary conditions using DD, and using an alternating method between mesh and physical solves, we propose two main approaches to solve the advection-diffusion equation (4.2.5) with periodic boundary conditions in parallel, using equidistribution. The main challenge in proposing the iteration lies in combining mesh equidistribution with the periodic boundary conditions in the physical PDE. We see in the periodic iteration (5.2.4) that the authors of [46] use fixed grids r_1 and r_2 throughout. If we add in the layer of equidistribution, these grids will change at each time step. In the DD mesh solve, recall that we partition the computational domain $\xi = [0, 1]$ into $\xi_1 = [\alpha_1, \beta_1]$ and $\xi_2 = [\alpha_2, \beta_2]$, where $\alpha_1 = 0, \beta_2 = 1$, and solve for r_1 and r_2 accordingly. With a mesh solve occurring at each time step, r_1 and r_2 will be modified at each time step k. This means that while the computational subdomains are fixed, the resulting mesh is not. This creates an issue with the setup of the periodic DD iteration (5.2.3), as δ and $\frac{b}{2}$ are not guaranteed to be node values in the updated subdomains r_1 and r_2.

Consider subdomains r_1 and r_2 given by

$$\begin{aligned} r_1 &= (r_1(0), \ldots, r_1(\beta_1))^T, \\ &= (r_{1,0}, r_{1,1}, \ldots, r_{1,m-1}, r_{1,m})^T, \end{aligned}$$

and

$$\begin{aligned} r_2 &= (r_2(\alpha_2), \ldots, r_2(1), r_2(1+\tilde{\delta}))^T, \\ &= (r_{2,0}, r_{2,1}, \ldots, r_{2,m-1}, r_{2,m})^T. \end{aligned}$$

Let

$$r_{1,0} = 0, \quad r_{2,m-1} = b, \quad \text{and} \quad r_{2,m} = b + \delta. \tag{5.2.5}$$

To numerically solve the mesh BVP given by

$$\frac{d}{d\xi}\left(F(r(\xi))\frac{dr(\xi)}{d\xi}\right) = 0, \qquad r(0) = 0,\ \ r(1) = b,$$

we fix $r_{2,m} = b + \delta$ and complete a DD mesh solve on $r_1 = (r_{1,0}, \dots, r_{1,m})^T$ and $r_2 = (r_{2,0}, \dots, r_{2,m-1})^T$, omitting the last entry of r_2. This allows us to equidistribute from $r = 0$ to $r = b$ and follow the single domain iteration.

Suppose the computational domain ξ is decomposed into $S = 2$ subdomains such that $\Omega_1 = [0, \beta]$, $\Omega_2 = [\alpha, 1]$ with $\alpha < \beta$. The mesh DD iteration is as follows: For each time step $k = 1, 2, \dots,$ for $n = 1, 2, \dots,$ solve

$$\frac{d}{d\xi}\left(F(r_1^{(n)})\frac{dr_1^{(n)}}{d\xi}\right) = 0, \xi \in \Omega_1, \qquad \frac{d}{d\xi}\left(F(r_2^{(n)})\frac{dr_2^{(n)}}{d\xi}\right) = 0, \xi \in \Omega_2, \tag{5.2.6}$$

$$r_1^{(n)}(0) = 0, \qquad r_2^{(n)}(\alpha) = r_1^{(n)}(\alpha), \tag{5.2.7}$$

$$r_1^{(n)}(\beta)) = r_2^{(n-1)}(\beta), \qquad r_2^{(n)}(1) = b, \tag{5.2.8}$$

The first main approach to enforce the periodic boundary conditions is given in iteration (5.2.9). The physical DD iteration is as follows: for each time step $k = 1, 2, \dots,$ for $n = 1, 2, \dots,$ solve

$$U_1^{(n)} = u_1^{(k-1)} + \Delta t\mathcal{L}U_1^{(n)}, \quad r \in \Omega_1, \qquad U_2^{(n)} = U_1^{(k-1)} + \Delta t\mathcal{L}U_2^{(n)}, \quad r \in \Omega_2, \tag{5.2.9}$$

$$U_1^{(n)}(0) = U_2^{(n-1)}(b), \qquad U_2^{(n)}(r_2^{(k-1)}(\alpha_2)) = U_1^{(n-1)}(r_1^{(k-1)}(\alpha_2)),$$

$$U_1^{(n)}(r_1^{(k-1)}(\beta_1)) = U_2^{(n-1)}(r_2^{(k-1)}(\beta_1)) \qquad U_2^{(n)}(b+\delta) = U_1^{(n-1)}(I_{r_1}(\delta)),$$

where the linear operator $\mathcal{L}$ is given by a centered finite difference approximation to (4.2.5). We note that $r_1^{(k-1)}$ and $r_2^{(k-1)}$ are converged mesh iterations from the previous time step, that is $r_2^{(k-1)}(\alpha_2) = r_1^{(k-1)}(\alpha_2)$ and $r_1^{(k-1)}(\beta_1) = r_2^{(k-1)}(\beta_1)$.

Here, $U_1^{(n-1)}(I_{r_1}(\delta))$ denotes the interpolant of the $U_1^{(n-1)}$ vector evaluated at $r = \delta$. To compute this, we construct a cubic spline of $U_1^{(n-1)}$ on $r_1^{(k-1)}$, and evaluate the spline at δ. We note that an interpolant such as $U_2^{(n-1)}(I_{r_2}(b))$ is not needed as to follow the single domain mesh BVP, as we have fixed the node $r_{2,m-1} = b$.

We see that the iteration (5.2.9) allows us to complete a mesh solve over the entire computational domain while obeying the periodic boundary conditions required by the periodic boundary conditions of the physical problem.

Another method used to solve the time-dependent problem with domain decomposition and equidistribution is the iteration (5.2.10). We note that in iteration (5.2.10), no interpolation is needed. Instead of interpolating to obtain the value of $U_1(\delta)$, we fix the node $r_{1,1} = \delta$, in addition to fixing the nodes in (5.2.5). For time step $k = 1, 2, \dots,$ for $n = 1, 2, \dots,$ solve

$$U_1^{(n)} = U_1^{(k-1)} + \Delta t \mathcal{L} U_1^{(n)}, \quad r \in \Omega_1, \qquad U_2^{(n)} = U_1^{(k-1)} + \Delta t \mathcal{L} U_2^{(n)}, \quad r \in \Omega_2, \tag{5.2.10}$$

$$U_1^{(n)}(a) = U_2^{(n-1)}(b), \qquad U_2^{(n)}(r_2^{(k-1)}(\alpha_2)) = U_1^{(n-1)}(r_1^{(k-1)}(\alpha_2)),$$
$$U_1^{(n)}(r_1^{(k-1)}(\beta_1)) = U_2^{(n-1)}(r_2^{(k-1)}(\beta_1)) \qquad U_2^{(n)}(b+\delta) = U_1^{(n-1)}(\delta),$$

This method has a clear disadvantage in that it will not equidistribute in certain areas of the domain. If the mesh nodes need to be concentrated in the interval $[0, \delta]$, the equidistribution will not produce the desired mesh and we could miss important features of the solution. Keeping the mesh uniform in certain areas could result in the same problems we obtain with an overall uniform mesh. Generally, we recommend against this method unless one is sure that the nodes need not be concentrated in the interval $r \in [0, \delta]$.

Before we continue with the algorithm, we provide notation. Since (5.2.9) and (5.2.10) turn the periodic PDE into a Dirichlet BVP on both subdomains, we define a physical time solve accordingly for our algorithm. Consider a general domain $\tilde{\Omega} = (\tilde{r}_0, \tilde{r}_1, \dots, \tilde{r}_m)^T$ and $\tilde{r}_{i-1} < \tilde{r}_i < \tilde{r}_{i+1}$. This $\tilde{\Omega}$ could be the single domain $\tilde{\Omega} = r$ or a Dirichlet subdomain $\tilde{\Omega} = r_1$ or $\tilde{\Omega} = r_2$. With boundary values $\mathcal{B}_1, \mathcal{B}_2$, time increment Δt, and time-independent source term $u^{(k-1)}$, we approximate the solution to the Dirichlet problem

$$u_t = (\psi(r)\psi_r(r) + \psi(r))u_r + \psi(r)\psi(r)u_{rr} + f(r,t), \quad r \in \tilde{\Omega}, \ u(r_0) = \mathcal{B}_1, \ u(r_m) = \mathcal{B}_2, \tag{5.2.11}$$

using an implicit backward Euler scheme at each time step. This gives the linear problem: at time step k, for $i = 1, \ldots, m-1$, and suppressing the time iteration k,

$$U_i = U^{(k-1)} + \Delta t\Bigg(A_i \frac{h_{i-1}^2(U_{i+1} - U_i) + h_i^2(U_i - U_{i-1})}{h_{i-1}h_i(h_{i-1} + h_i)} \\ + B_i \frac{2[h_{i-1}(U_{i+1} - U_i) - h_i(U_i - U_{i-1})]}{h_{i-1}h_i(h_{i-1} + h_i)} + f_i \Bigg), \tag{5.2.12}$$

where $U_0 = \mathcal{B}_1, U_m = \mathcal{B}_2$. Here, $A_i = \psi(r_i)\psi_r(r_i) + \psi(r_i), B_i = \psi(r_i)\psi(r_i)$, and $f_i = f(r_i, t^{(k-1)})$.

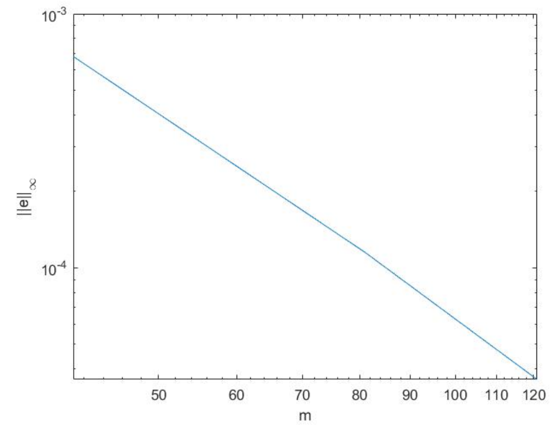
Approximating the solution to (5.2.11) with the above discretization will be called a "DD physical time solve", which we will label as a "$P_{dd,t}$" solve on $\tilde{\Omega}$ moving forward. We note that this discretization arises from the centered difference approximation, but one could similarly employ an upwind approximation.

We begin with an initial guess $u^{(0)}$ and choose $U_i^{(0)} = u^0|_{\Omega_i}$. We also choose an initial guess of a uniform mesh on each Ω_i, that is $r_i^{(0)}$ is uniform in ξ. The proposed classical Schwarz iterative procedure is given in Algorithm 7. We note that unless Δt is chosen to be small enough so that the time error is negligible with respect to the chosen spatial tolerance, the tolerance imposed in Algorithm 7 will not actually control the error of the physical solution on each time step. Here, we initialize $\Delta t = \frac{\Delta r^2}{2}$ for the initial uniform mesh r and keep Δt constant, as mentioned in Section 4.2.

We now provide numerical results for Algorithm 7 which uses (5.2.9). Here, the error $||e||_\infty$ is defined as the difference between the numerical DD solution provided by the $(M_\infty(P_{dd,t})_\infty)_\nu$ method and the numerical single domain solution for the MP_t method. To compare the solutions, we remove the last entry of $r_2^{(k)}$ and $U_2^{(k)}$, as these entries "overlap" on the periodic domain. These results are shown in Figure 5.9. We obtain $O(h^{2.7})$ for the mesh error and $O(h^{2.3})$ for the physical solution error.

Algorithm 7: $(M_\infty(P_{dd,t})_\infty)_\nu$ Method

```
Use the derivatives of the curve $\mathbf{x}(r)$ and (4.1.4) to evaluate the function $\psi(r)$;
Choose a uniform initial mesh $r^{(0)}$;
Use the initial condition to obtain $U^{(0)}$;
for $k = 0, 1, \ldots,$ do
    set $t^{(k+1)} = t^{(k)} + \Delta t$ ;
    initialize $r_i^{(0)} = r_i^{(k-1)}$ for $i = 1, \ldots, S$ ;
    for $c = 1, 2, \ldots,$ do
        for $i = 1, 2$ do
            Complete a mesh solve on the computational coordinate domain
             $\Omega_i = [\alpha_i, \beta_i]$ with boundary values
            $\begin{cases} r_{i,0} = 0, & r_{i,m} = r_{i+1}^{(c-1)}(\beta_i), \quad \text{for } i = 1, \\ r_{i,0} = r_{i-1}^{(c-1)}(\alpha_i), & r_{i,m-1} = b, \quad \text{for } i = 2, \end{cases}$
            with $r_{i,m} = b + \delta$ to obtain $r_i^{(c)}$ ;
        end
        if $\max(||r_1^{(c)} - r_1^{(c-1)}||_\infty, ||r_2^{(c)} - r_2^{(c-1)}||_\infty) < tol/10$ then
            BREAK LOOP;
        end
    end
    Result: $r_i^{(k)}$ for $i = 1, 2$
    for $c = 1, 2, \ldots,$ do
        Complete a DD physical time solve on $r_1 = r_1^{(k)}$ and $r_2 = r_2^{(k)}$ using
         iteration (5.2.9) and discretizing using (5.2.12) to obtain $U_1^{(c)}$ on $r_1$
         and $U_2^{(c)}$ on $r_2$ ;
        if $\max(||U_1^{(c)} - U_1^{(c-1)}||_\infty, ||U_2^{(c)} - U_2^{(c-1)}||_\infty) < tol/10$ then
            BREAK LOOP;
        end
    end
    Result: $U_i^{(k)}$ for $k = 1, 2$
end
```



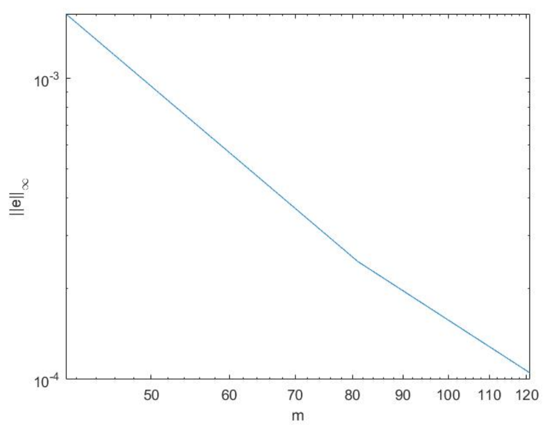


Figure 5.9: A plot of the infinity norm of the error between the $(M_\infty(P_{dd,t})_\infty)_\nu$ numerical solution and the single domain MP_t solution used to solve (5.2.4) for a varying numbers of nodes, m, at $t = 0.001$. The DD error is shown for the mesh $r^{(k)}$ (left) and the physical solution $U^{(k)}$ (right). Here, we use monitor function (2.4.4) with weights $\omega = 1$ and $\omega_u = 1$.

An important note is that these numerical results vary depending on the monitor function used in equidistribution. This discrepancy may occur for many reasons. As we are producing (and taking the derivatives of) a spline on a non-uniform mesh, the accuracy of the spline will depend on $\max_i h_i$. If the grid spacing is larger in certain areas, it can affect the accuracy of the spline representation of the monitor function. For example, if we change the weights in the monitor function (2.4.4) to $\omega = 0.1$ and $\omega_u = 0.1$, we obtain $O(h^{4.4})$ for the mesh error and $O(h^{2.3})$ for the physical solution error. This is shown in Figure 5.10. We conclude that this varying order of accuracy depending on the monitor function is due to the interpolation. We conclude this because in practice, when we choose a known analytical solution $u(r,t)$ such that the monitor function $F(r, u_r(r,t))$ can be analytically determined at each time step (without interpolation), the DD mesh converges to the single domain mesh $r^{(k)}$ within the chosen DD tolerance.

This is the same result that occurred in the static boundary layer case in Figure 5.4 in Section 5.1. Generally, since the physical solution is computed on the equidistributed mesh, the order of accuracy from the single domain solution of the physical DD solution will be limited by the accuracy of the mesh.

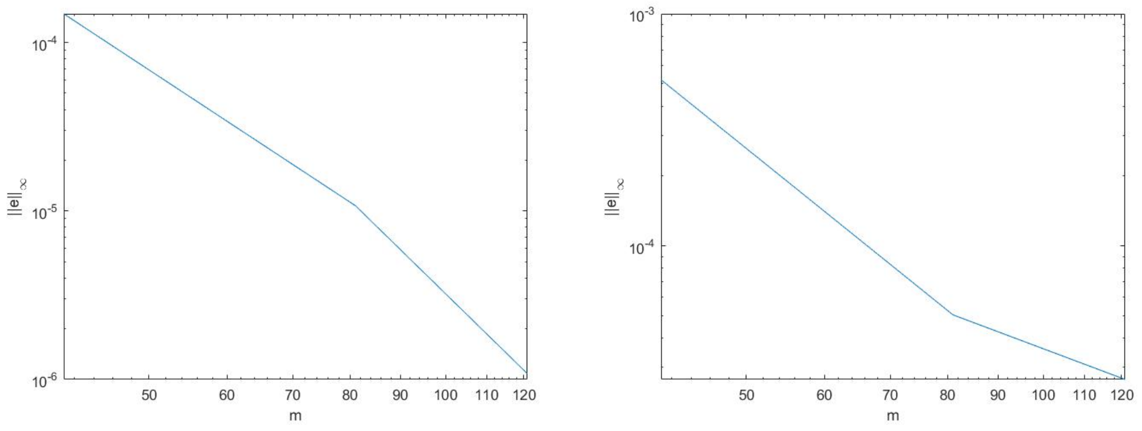


Figure 5.10: A plot of the infinity norm of the error between the $(M_\infty(P_{dd,t})_\infty)_\nu$ solution and the single domain MP_t solution used to solve (5.2.4) for a varying numbers of nodes, m, at $t = 0.001$. The DD error is shown for the mesh $r^{(k)}$ (left) and the physical solution $U^{(k)}$ (right). Here, we use monitor function (2.4.4) with weights $\omega = 0.1$ and $\omega_u = 0.1$.

Recall that from Tables 4.1 and 4.2 in Section 4.2, for this example problem we obtained low interpolation errors for the monitor functions (2.4.4) and (2.4.19). In practice, choosing a monitor function that uses the second derivative of u, such as monitor function (2.4.19), may have an effect on the order of the DD error, as we are taking the second derivative of a spline. Numerical DD error results from the single domain solution are shown in Figure 5.11. We see that while we are still obtaining convergence to the single domain solution, we are obtaining relatively larger errors when compared to Figures 5.9 and 5.10.

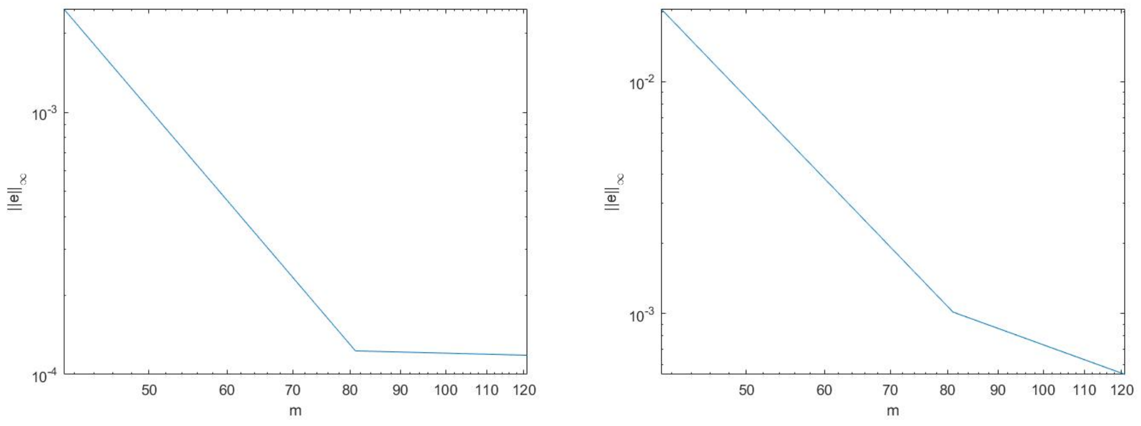


Figure 5.11: A plot of the infinity norm of the error between the $(M_\infty(P_{dd,t})_\infty)_\nu$ solution and the single domain MP_t solution used to solve (5.2.4) for a varying numbers of nodes, m, at $t = 0.001$. Here, we use monitor function (2.4.19) with weights $\omega = 0.1$ and $\omega_u = 0.1$.

There are many moving pieces to this iteration. Numerical errors occur from the time discretization, numerical discretization, interpolation methods, etc. However, our alternating DD iteration given by Algorithm 7 generally converges to the numerical solution given by the single domain Algorithm 3. Algorithm 3 produced a solution with a lower interpolation error when compared to a coarse uniform grid of the same size.

Chapter 6

Summary and Future Work

In this thesis we have explored the concept of equidistribution applied to parametric curves. In Chapter 2, we focused on single domain equidistribution, including the line and curve case. To generate a grid on a parametric curve $(x(r), y(r))$ in $\mathbb{R}^2$, we used the curve features (arc-length and curvature) to equidistribute the nodes. We also briefly discussed equidistribution on a time-dependent curve, where we solved a mesh BVP at each time step to compute new mesh at each time step. To equidistribute a known function $u(r)$ posed on the curve, we tested using multiple monitor functions, comparing them by their interpolation errors. A main takeaway is that when equidistributing PDEs posed on curves, it is beneficial to choose a monitor function that equidistributes using both the curve and solution features of a given curve.

Chapter 3 focused on domain decomposition methods for equidistribution on a curve. Parallel computing allows us to reduce the additional computational cost that occurs with r-refinement. We briefly explained the general parallel classical Schwarz and optimized Schwarz iterations, supplying and citing previous analytical convergence results. We also carried out a numerical experiment involving the contraction factor of the optimized Schwarz iteration, comparing the expected optimal p value to the experimental optimal p value. This value depends on the curve-dependent bounds of the monitor function $F(r)$ on the number of mesh points m. We then specifically stated the parallel Schwarz iterations for the equidistributing mesh on a curve. We also supplied analytical convergence results for these methods, slightly modifying an already established convergence result for both DD methods. As is common in the literature, we concluded that when an appropriate choice of the Robin parameter, ρ, is used, the optimized Schwarz iteration provides faster convergence than classical

Schwarz. We remark that in this thesis, we have focused on leveraging multiple CPUs (Central Processing Units) with domain decomposition. An interesting extension for future work would be to see how GPU (Graphic Processing Unit) parallelism can be applied. GPU parallelism, particularly a domain decomposition method on a low cost GPU cluster, has been discussed in [43] and extended to a larger cluster in [2].

Chapters 4 and 5 involved equidistribution of solutions of differential equations on curves. Chapter 4 explored equidistribution of these differential equations on a single domain. When an equation is posed on a parametric curve, the common differential operators such as the surface gradient and Laplace-Beltrami operators reduce to the first and second derivatives multiplied by additional curve-based coefficients. In the both the static boundary layer case and the time-dependent PDE case, we employed an alternating method. Specifically in the static boundary layer case, this alternating method begins with a uniform mesh r_0, on which we solve for U_0. Using this approximation in the monitor function, we solve for r_1, and so on. We supplied numerical results showing that for a small number of nodes, equidistribution results in a mesh that much more accurately resolves the solution when compared to a uniform mesh. Chapter 4 also introduced single domain equidistribution of time-dependent PDEs posed on curves. In this case, we also used periodic boundary conditions to show how one would deal with this problem posed on a closed curve. To solve the time-dependent problem, we begin with the initial condition and impose an alternating iteration, completing a mesh solve and physical solve at each time step. The problem was discretized using implicit backward Euler with a centered difference spatial discretization, but any stable time discretization could be used. Chapter 5 introduced multi-domain approaches to the problems discussed in Chapter 4. In the static boundary layer case, the outer loop becomes the alternating loop, and we complete either full or partial DD solves at each alternating iteration. We concluded that generally, the partial solves method was able to reduce the total number of linear solves without sacrificing the accuracy. In the time-dependent case, we proceed with solving a sequence of elliptic equations at each time step. That is, the time loop remains the outermost loop and each subdomain problem becomes a backward Euler step. To obey the periodic boundary conditions, we follow the method proposed by [46] with slight modification, giving Dirichlet transmission conditions on each subdomain. We concluded that as expected, the DD iteration converges to the single domain numerical solution. In Chapter 5, we chose to do domain decomposition in the r variable;

that is, we directly partitioned the $r \in [0, b]$ interval into equally sized subdomains and computed the physical solution on r_1 and r_2. Another technique would be to do domain decomposition in the ξ variable; this would include considering the physical solution $u(r(\xi), t)$ and recomputing the derivatives of u by the chain rule. This would give $u_\xi = \frac{u(r(\xi),t)}{dr}\frac{dr}{d\xi}$ and so on. While Chapter 5 focused on classical Schwarz as the DD method, one could easily adjust the transmission conditions to optimized Schwarz if desired.

We now summarize the contributions provided in this thesis. The main purpose of this thesis was to explore the equidistribution of a function $u(r)$, in the case where $u(r)$ is known in addition to the case where $u(r)$ is the solution of a differential equation. For a function $u(r)$ posed on $\mathbf{x}(r)$, we explored monitor functions which take in a mixture of curve and function features, which to our knowledge has not been done before. Our numerical results suggest that choosing a monitor function which considers both the curve and function features will minimize the interpolation error from the fine grid solution. We have proposed multiple monitor functions that fit this criteria.

Additionally, we contribute iterations to solve differential equations posed on curves combined with equidistribution. We also add in the layer of domain decomposition, proposing a classical Schwarz iteration that could easily be adapted to an optimized Schwarz iteration. The author of [36] provided an alternating iteration on a single domain interval to combine mesh equidistribution with the solution of a static boundary layer problem. There has also been a considerable amount of work done using an alternating method to combine mesh equidistribution with the solution of a time-dependent PDE, see [31] for details. To our knowledge, this thesis is the first time this alternating technique has been used to solve differential equations posed on curves with equidistribution. There has been work done on combining moving mesh methods with domain decomposition; see [24, 18, 25]. However, to our knowledge, the concept of combining equidistribution and domain decomposition with an alternating iteration to solve differential equations on curves is novel. We provide numerical evidence that the single domain algorithm converges to the numerical solution on a uniform find grid. We also provide numerical evidence that the DD algorithm converges to the single domain solution.

We now discuss additional opportunities for future work. As mentioned in Chapter 2, we have focused on providing a proof of concept of r-refinement on curves, rather than providing the overall most efficient method. Future work providing a careful implementation and comparison of these adaptive methods versus a fine uniform mesh should be completed, where the overall efficiencies of the methods are compared. We note that the authors of [33] showed that on the plane with a finite element discretization, implementing hr-refinement can produce a lower overall computational time when compared to only adding additional nodes (h-refinement). There is no obvious reason that the same benefit would not hold for r-refinement on a curve. It is also important to note that in Chapters 2 and 4, our goal is not to optimize the choice of monitor function or its parameters; we leave this for future work. We have provided sensible monitor function selections and shown that even among this small selection with no attempt of optimization, there exist monitor functions that produce a reduction in interpolation error when compared to a uniform mesh. Future work attempting to choose an optimal monitor function should be pursued. Additionally, as discussed in Chapter 2, there are other methods to compare the equidistributed solution to the uniform solution that should be explored.

An important aspect of future work revolves around theoretical results. Throughout this thesis we have provided mainly numerical results, showing convergence results numerically through error plots, tables showing interpolation errors, and so on. Future analytical work on a single domain would include error bounds and proof of convergence. As well, throughout this thesis we have not focused on choosing the "best" discretization. Instead we have generally chosen centered finite differences as a spatial discretization coupled with a backward Euler time discretization for the time-dependent problem. Future work could involve studying different discretizations such as finite element methods or a higher order Runge-Kutta time discretization. We also remark that depending on the time-dependent PDE, certain discretizations should be used to maintain stability. For example, the advection equation generally loses stability with a centered difference scheme, and is often discretized using an upwind method.

Additionally, an immediate extension of the work produced in this thesis would be to extend the classical Schwarz algorithms provided in Chapter 5 to the optimized Schwarz case. There is no immediate reason why this extension would not follow in a straightforward manner. Additionally, a fundamental extension to Chapters 3, 4 and 5 would be to consider the case of time-dependent curves. While we have introduced the framework of equidistribution on time-dependent curves in Chapter 2, all of the work in Chapters 3, 4, and 5 considers the case of static curves.

Another natural extension of the work provided in this thesis involves equidistribution on surfaces. As we have discussed parametric curves throughout this thesis, the natural progression is PDEs posed on a parametric surface of the form

$$\mathbf{x}(\mathbf{r}) : S^2 \to \mathbb{R}^3, \quad \mathbf{x}(\mathbf{r}) = (x_1(\mathbf{r}), x_r(\mathbf{r}), x_3(\mathbf{r}))^T, \quad \mathbf{r} = (r_1, r_2)^T.$$

If the parameterization of the surface is known, efficient methods can be formed to solve a PDE posed on the surface [11]. There has been an abundance of research on the topic of generating grids on surfaces; a thorough reference for grid generation on parametric surfaces can be found in [39]. We remark that parametric representation is only one of many ways to represent a surface. Future work can also involve triangulated surfaces, as triangulation is a common technique used to represent 3D surfaces. Additionally, a recent method used to solve PDEs on surfaces is the Closest Point method (CPM) [47]. The CPM is an embedding method, representing a surface using a function that maps points to their closest point on the surface. We note that this method can also be applied to 2D curves as well. In [41], the authors used domain decomposition as both a preconditioner and solver to solve the positive Helmholtz equation posed on a surface with the Closest Point method paired with finite difference methods.

A more recent method to solve time-dependent PDEs in parallel is called Schwarz Waveform relaxation, proposed by Gander et. al [12] and independently by Giladi [23]. In SWR, we decompose the spatial domain and solve the time-dependent PDE on the entire time interval on each subdomain. For a general linear time-dependent problem $u_t = \mathcal{L}u$, $x \in \Omega$, the two subdomain iteration is given by

$$\begin{aligned} u_{1,t}^{(n)} &= \mathcal{L}u_1^{(n)}, \quad (x,t) \in \Omega_1 \times [0,T], \\ \mathcal{B}_1 u_1^{(n)} &= \mathcal{B}_1 u_2^{(n-1)}, \quad (x,t) \in \Omega_1 \times [0,T], \\ u_{2,t}^{(n)} &= \mathcal{L}u_2^{(n)}, \quad (x,t) \in \Omega_2 \times [0,T], \\ \mathcal{B}_2 u_2^{(n)} &= \mathcal{B}_2 u_1^{(n-1)}, \quad (x,t) \in \Omega_2 \times [0,T], \end{aligned} \tag{6.0.1}$$

subject to given boundary and transmission conditions. We note in (6.0.1), the computed solution on each subdomain is not exchanged at each time step. We can view the time loop as the "outer" loop and the DD iteration as the "inner" loop. The subdomains only need to communicate at the end of the time interval. However, a larger amount of data will need to be communicated at each DD iteration. SWR allows us to use varying time steps during the iteration. Convergence results have been established for multiple well known PDEs with both Dirichlet and Robin transmission conditions, see [21, 4, 3, 15, 16, 17, 20, 22]. We remark that the SWR method can also be extended to $n > 2$ subdomains. Although we do not pursue this approach for the time-dependent PDEs on curves with equidistribution, we note that a first step in this direction was provided in [24, 25].

Bibliography

[1] D. Adalsteinsson and J. A. Sethian, *Transport and diffusion of material quantities on propagating interfaces via level set methods*, Journal of Computational Physics, 185 (2003), pp. 271–288.

[2] R. Babich, G. Shi, M. Clark, R. Brower, B. Joó, and S. Gottlieb, *Scaling lattice qcd beyond 100 GPUs*, in SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2011, pp. 1–11.

[3] D. Bennequin, M. J. Gander, and L. Halpern, *A homographic best approximation problem with application to optimized Schwarz waveform relaxation*, Mathematics of Computation, 78 (2009), pp. 185–223.

[4] M. Bjorhus, *On Domain Decomposition, Subdomain Iteration and Waveform Relaxation*, PhD thesis, Department of Mathematical Sciences, Norwegian Institute of Technology, 1995.

[5] X.-C. Cai, *Additive Schwarz algorithms for parabolic convection-diffusion equations*, Numerische Mathematik, 60 (1991), pp. 41–61.

[6] ——, *Multiplicative Schwarz methods for parabolic problems*, SIAM Journal on Scientific Computing, 15 (1994), pp. 587–603.

[7] C. De Boor, *Good approximation by splines with variable knots. ii*, in Conference on the Numerical Solution of Differential Equations, Springer, 1974, pp. 12–20.

[8] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015.

[9] T. Eiter and H. Mannila, *Computing discrete Fréchet distance*, tech. rep., Citeseer, 1994.

[10] C. M. Elliott and B. Stinner, *Modeling and computation of two phase geometric biomembranes using surface finite elements*, Journal of Computational Physics, 229 (2010), pp. 6585–6612.

[11] M. S. Floater and K. Hormann, *Surface parameterization: a tutorial and survey*, Advances in Multiresolution for Geometric Modelling, (2005), pp. 157–186.

[12] M. J. Gander, *A waveform relaxation algorithm with overlapping splitting for reaction diffusion equations*, Numerical Linear Algebra with Applications, 6 (1999), pp. 125–145.

[13] ——, *Optimized Schwarz methods*, SIAM Journal on Numerical Analysis, 44 (2006), pp. 699–731.

[14] ——, *Schwarz methods over the course of time*, Electronic Transactions on Numerical Analysis, 31 (2008), pp. 228–255.

[15] M. J. Gander and L. Halpern, *Absorbing boundary conditions for the wave equation and parallel computing*, Mathematics of Computation, 74 (2005), pp. 153–176.

[16] ——, *Optimized Schwarz waveform relaxation methods for advection reaction diffusion problems*, SIAM Journal on Numerical Analysis, 45 (2007), pp. 666–697.

[17] M. J. Gander, L. Halpern, and F. Nataf, *Optimal Schwarz waveform relaxation for the one dimensional wave equation*, SIAM Journal on Numerical Analysis, 41 (2003), pp. 1643–1681.

[18] M. J. Gander and R. D. Haynes, *Domain decomposition approaches for mesh generation via the equidistribution principle*, SIAM Journal on Numerical Analysis, 50 (2012), pp. 2111–2135.

[19] M. J. Gander, R. D. Haynes, and A. J. Howse, *Alternating and linearized alternating Schwarz methods for equidistributing grids*, in Domain Decomposition Methods in Science and Engineering XX, Springer, 2013, pp. 395–402.

[20] M. J. Gander and C. Rohde, *Overlapping Schwarz waveform relaxation for convection-dominated nonlinear conservation laws*, SIAM Journal on Scientific Computing, 27 (2005), pp. 415–439.

[21] M. J. Gander and A. M. Stuart, *Space-time continuous analysis of waveform relaxation for the heat equation*, SIAM Journal on Scientific Computing, 19 (1998), pp. 2014–2031.

[22] M. J. Gander and H. Zhao, *Overlapping Schwarz waveform relaxation for the heat equation in n dimensions*, BIT Numerical Mathematics, 42 (2002), pp. 779–795.

[23] E. Giladi and H. B. Keller, *Space-time domain decomposition for parabolic problems*, Numerische Mathematik, 93 (2002), pp. 279–313.

[24] R. D. Haynes, *The numerical solution of differential equations: grid selection for boundary value problems and adaptive time integration strategies*, PhD thesis, Theses (Dept. of Mathematics)/Simon Fraser University, 2003.

[25] R. D. Haynes, W. Huang, and R. D. Russell, *A moving mesh method for time—dependent problems based on Schwarz waveform relaxation*, in Domain Decomposition Methods in Science and Engineering XVII, Springer, 2008, pp. 229–236.

[26] R. D. Haynes and F. Kwok, *Discrete analysis of domain decomposition approaches for mesh generation via the equidistribution principle*, Mathematics of Computation, 86 (2017), pp. 233–273.

[27] N. J. Hicks, *Notes on differential geometry*, Van Nostrand Mathematical Studies, No. 3, D. Van Nostrand Co., Inc., Princeton, N.J.-Toronto-London, 1965.

[28] A. J. M. Howse, *Domain decomposition approaches for the generation of equidistributing grids*, Master's thesis, Memorial University of Newfoundland, 2013.

[29] W. Huang, Y. Ren, and R. D. Russell, *Moving mesh partial differential equations (MMPDEs) based on the equidistribution principle*, SIAM Journal on Numerical Analysis, 31 (1994), pp. 709–730.

[30] W. Huang, Y. Ren, R. D. Russell, et al., *Moving mesh methods based on moving mesh partial differential equations*, Journal of Computational Physics, 113 (1994), pp. 279–290.

[31] W. Huang and R. D. Russell, *Adaptive moving mesh methods*, vol. 174, Springer Science & Business Media, 2010.

[32] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, *Comparing images using the hausdorff distance*, IEEE Transactions on pattern analysis and machine intelligence, 15 (1993), pp. 850–863.

[33] H. Jahandari, S. MacLachlan, R. D. Haynes, and N. Madden, *Finite element modelling of geophysical electromagnetic data with goal-oriented hr-adaptivity.*, Computational Geosciences, 24 (2020).

[34] A. J. James and J. Lowengrub, *A surfactant-conserving volume-of-fluid method for interfacial flows with insoluble surfactant*, Journal of Computational Physics, 201 (2004), pp. 685–722.

[35] D. Kamilis, *Numerical methods for the PDEs on curves and surfaces*, Master's thesis, Umeå University, 2013.

[36] N. Kopteva, *Convergence theory of moving grid methods*, in Adaptive Computations: Theory and Algorithms, T. Tang and J. Xu, eds., Science Press, Beijing, 2007, ch. 4, pp. 159–210.

[37] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.

[38] P.-L. Lions, *On the Schwarz alternating method. II. Stochastic interpretation and order properties*, in Domain Decomposition Methods (Los Angeles, CA, 1988), SIAM, Philadelphia, PA, 1989, pp. 47–70.

[39] V. D. Liseikin, *Grid generation methods*, Scientific Computation, Springer, Cham, third ed., 2017.

[40] J. Lowengrub, J. Xu, A. Voigt, et al., *Surface phase separation and flow in a simple model of multicomponent drops and vesicles*, Fluid Dyn. Mater. Proc, 3 (2007), pp. 1–19.

[41] I. C. May, R. D. Haynes, and S. J. Ruuth, *Schwarz solvers and preconditioners for the closest point method*, SIAM Journal on Scientific Computing, 42 (2020), pp. A3584–A3609.

[42] J. D. Murray, *Mathematical biology: I. An introduction*, vol. 17, Springer Science & Business Media, 2007.

[43] Y. Osaki and K.-I. Ishikawa, *Domain decomposition method on GPU cluster*, arXiv:1011.3318, (2010).

[44] D. W. Peaceman and H. H. Rachford, Jr, *The numerical solution of parabolic and elliptic differential equations*, Journal of the Society for industrial and Applied Mathematics, 3 (1955), pp. 28–41.

[45] J. Pryce, *On the convergence of iterated remeshing*, IMA Journal of Numerical Analysis, 9 (1989), pp. 315–335.

[46] A. Qaddouri, L. Laayouni, S. Loisel, J. Côté, and M. J. Gander, *Optimized Schwarz methods with an overset grid for the shallow-water equations: preliminary results*, Applied Numerical Mathematics, 58 (2008), pp. 459–471.

[47] S. J. Ruuth and B. Merriman, *A simple embedding method for solving partial differential equations on surfaces*, Journal of Computational Physics, 227 (2008), pp. 1943–1961.

[48] H. A. Schwarz, *Ueber einen Grenzübergang durch alternirendes Verfahren*, Zürcher u. Furrer, 1870.

[49] Y. Seol and M.-C. Lai, *Spectrally accurate algorithm for points redistribution on closed curves*, SIAM Journal on Scientific Computing, 42 (2020), pp. A3030–A3054.

[50] J. M. Stockie, J. A. Mackenzie, and R. D. Russell, *A moving mesh method for one-dimensional hyperbolic conservation laws*, SIAM Journal on Scientific Computing, 22 (2001), pp. 1791–1813.

[51] X. Xu, W. Huang, R. Russell, and J. Williams, *Convergence of de Boor's algorithm for the generation of equidistributing meshes*, IMA Journal of Numerical Analysis, 31 (2011), pp. 580–596.